SECOND EDITION

# PC INTERRUPTS

## A PROGRAMMER'S REFERENCE TO BIOS, DOS, AND THIRD-PARTY CALLS

RALF BROWN & JIM KYLE

DP

SECOND EDITION

# PC INTERRUPTS

## A PROGRAMMER'S REFERENCE TO BIOS, DOS, AND THIRD-PARTY CALLS

Ralf Brown and Jim Kyle

Addison-Wesley Publishing Company

Reading, Massachusetts    Menlo Park, California    New York
Don Mills, Ontario    Wokingham, England    Amsterdam    Bonn
Sydney    Singapore    Tokyo    Madrid    San Juan
Paris    Seoul    Milan    Mexico City    Taipei

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial capital letters.

The authors and publishers have taken care in preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

ISBN 0-201-62485-0

QA
76.6
.B7734
1994

Copyright © 1994 by Ralf Brown and Jim Kyle

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

Production Editor: Andrew Williams
Set in 10 point Galliard by Benchmark Productions, Inc.

1 2 3 4 5 6 7 8 9-BAH-9796959493
*First Printing, December 1993*

Addison-Wesley books are available for bulk purchases by corporations, institutions, and other organizations. For more information please contact the Corporate, Government, and Special Sales Department at (617) 944-3700 x2915.

# *Table of Contents*

**CHAPTER 1 Introduction** **1**
Why This Book Exists 1, Interrupts and the '86 CPU Family 2, Some Words of Caution 2, Sample Entry 3, About The Authors 4, Acknowledgments 4,

**CHAPTER 2 Organization and Master Interrupt List** **7**

**CHAPTER 3 Hardware Interrupts** **21**
CPU-generated Interrupts 21, External Hardware Interrupts 25

**CHAPTER 4 ROM BIOS** **31**

**CHAPTER 5 Expansion Bus BIOSes** **64**
EISA System ROM 64, Intel PCI BIOS 65, PCMCIA Socket Services 68

**CHAPTER 6 Japanese ROM BIOS Extensions** **77**
AX PC (Japanese) 77, NEC 80,

**CHAPTER 7 Vendor-Specific ROM BIOS Extensions** **83**
Acorn BBC Master 512 83, Amstrad 87,
Atari 89, AT&T 89, Compaq 90, Corona 94, Hewlett-Packard 94,
Phoenix 96, Tandy 96, Victor 96, Zenith 96

**CHAPTER 8 Video** **99**
Standard BIOS Calls 99, Vendor-Specific Extensions 138, EGA Register Interface Library 154, Hercules GRAFIX 157, UltraVision 159, Miscellaneous Display Drivers 163

**CHAPTER 9 SuperVGA** **183**
VESA SuperVGA BIOS Extensions 183, SOLLEX SuperVGA Extensions 188, Cirrus Logic 194, Trident Video BIOS 197, Tseng Video BIOS 197, OPTIMA/ET-3000 Zoom TSR 198, Miscellaneous 201

**CHAPTER 10 Low-Level Disk I/O** **203**
Standard BIOS 203, ESDI Controllers 212, Vendor-Specific Extensions 214, RAMdisks 218

**CHAPTER 11 SCSI Device I/O** **221**
ASPI 221, Common Access Method 224, CMC International
TARGA.DEV 230, Future Domain 234, SCSILink 240, SDLP 242,
Miscellaneous 243

**CHAPTER 12 Serial I/O** **247**
Standard BIOS 247, Digiboard 250, FOSSIL 253, MBBIOS 264, AVATAR Serial Dispatcher 266, COURIERS.COM 266, IBM/Yale EBIOS 268, TSRCOMM 269, Miscellaneous 271

**CHAPTER 13 Keyboard Enhancers** 275
Anarkey 275, DOSKEY 277, PCED 278, Phantom of the Keyboard 279, Miscellaneous 280
**CHAPTER 14 Screen Blankers** 287
After Dark 287, Blank-It 289, Explosiv 291, Miscellaneous 293
**CHAPTER 15 Printer Services and Print Spoolers** 295
Standard BIOS 295, Frieze 296, INSET 298, NorthNet Jetstream 300, Postscript converters 301, Print Spoolers—Disk Spool II 305, Print Spoolers—PCSpool 310, Print Spoolers—PRINT 312, Printer Redirectors 316, Miscellaneous 317
**CHAPTER 16 Disk Compression** 321
DIET 321, DoubleSpace 324, Microsoft Real-Time Compression Interface 328, NewSpace 329, Stacker 331, SuperStor 332, Miscellaneous 333
**CHAPTER 17 Disk Caches** 337
FAST! 337, HyperDisk 339, NOW! 341, PC-Cache 343, QuickCache II 344, SmartDrive 350, Super PC-Kwik 354, Miscellaneous 360
**CHAPTER 18 Virtual DMA Specification** 365
**CHAPTER 19 Advanced Power Management Specification** 371
**CHAPTER 20 Hewlett-Packard HP 95LX System Services** 379
**CHAPTER 21 Mouse Support** 437
PS/2 Pointing Device 437, Microsoft Mouse 439, Mouse Systems 453, Logitech 457, Smooth Mouse Driver 461, Quick Mouse Reset 462, Miscellaneous 463
**CHAPTER 22 Sound and Sound Boards** 465
Ad Lib 465, SoundBlaster 467, PCM driver 469, Media Vision 471, Tandy 477, SND 478, RESPLAY 479, IBMSND driver 480, FARTBELL 483, Speech 483
**CHAPTER 23 Screen Readers and Magnifiers** 485
AccessDOS 485, ASAP 485, AutoBraille 486, MAGic 488, Tinytalk Personal 490
**CHAPTER 24 MS-DOS Kernel** 493
INT 21h Services 495, INT 2Fh Services 582, Other Services 596
**CHAPTER 25 European MS-DOS 4.00** 603
Kernel 603, POPUP Utility 610
**CHAPTER 26 MS-DOS Compatibles** 613
Digital Research / Novell DOSes 613, OS/2 621, Miscellaneous 627
**CHAPTER 27 MS-DOS Utilities** 629
ANSI.SYS 629, APPEND 630, ASSIGN 632, Critical Error Handler 632, DISPLAY.SYS 633, GRAFTABL 634, GRAPHICS 635, KEYB 635, NLSFUNC 637, SHARE 638, XMA2EMS 640, Miscellaneous 641

**CHAPTER 28 COMMAND.COM and Replacements** **643**
COMMAND.COM 643, 4DOS / NDOS 645
**CHAPTER 29 Virtual Control Program Interface (VCPI)** **649**
**CHAPTER 30 DOS Protected-Mode Interface (DPMI)** **653**
**CHAPTER 31 DOS Protected Mode Services (DPMS)** **679**
**CHAPTER 32 Cloaking** **683**
**CHAPTER 33 Phar Lap and FlashTek DOS Extenders** **689**
Phar Lap 689, FlashTek Extensions 702
**CHAPTER 34 OS/286 and OS/386** **705**
**CHAPTER 35 Rational Systems DOS Extenders** **709**
**CHAPTER 36 Borland DOS Extenders** **715**
DPMILOAD 715, RTM 725, TKERNEL 726
**CHAPTER 37 Other DOS Extenders** **733**
GO32 733, DBOS (University of Salford Extender) 734, Miscellaneous 735
**CHAPTER 38 EMS and XMS** **737**
EMS 737, Extended Memory Specification (XMS) 746
**CHAPTER 39 Quarterdeck QEMM-386** **753**
**CHAPTER 40 Qualitas 386MAX** **769**
**CHAPTER 41 Helix Netroom** **783**
**CHAPTER 42 Other Memory Managers** **795**
Common Calls 795, Microsoft HIMEM and EMM386 797, Compaq CEMM 799, The Last Byte 800, Nanosoft MD386 802, Miscellaneous 803
**CHAPTER 43 Microsoft Windows** **805**
Windows Interrupt Calls 805, Virtual Device Drivers 813, Video Support 816, WINOLDAP 817, WINX 821
**CHAPTER 44 TopView, DESQView, and DESQView/X** **823**
TopView 823, DESQview 856, DESQview/X 874, DESQview External Device Interface 879, Miscellaneous 883
**CHAPTER 45 MultiDOS Plus** **887**
Multitasker 887, I/O Drivers 896
**CHAPTER 46 VMiX** **901**
**CHAPTER 47 DR-DOS TaskMAX** **911**
**CHAPTER 48 DR Multiuser DOS** **917**
**CHAPTER 49 Other Multitaskers/Task Switchers** **929**
Back and Forth 929, Cswitch 930, DOSSHELL 933, DOS Task Switcher 935, DoubleDOS 939, Helix Software Headroom 943, OmniView 945, PC-MOS/386 946, PC Tools CPTASK 960, Software Carousel 962, SWELL (Swapping Shell) 965, Switch-It 966, Miscellaneous 969
**CHAPTER 50 Alternate Multiplex Interrupt** **971**
**CHAPTER 51 TSR Libraries** **977**
**CHAPTER 52 FastGraph/Light** **987**
**CHAPTER 53 STSC APL*Plus/PC** **1013**
**CHAPTER 54 Programming Language Runtime Support** **1019**

**CHAPTER 55 Intel Image Processing Interface** **1031**

**CHAPTER 56 System Profilers** **1035**

Microsoft Profiler 1035, SYS_PROF 1037

**CHAPTER 57 Debuggers and Debugging Tools** **1039**

CAPDOS 1039, COMTROL Hostess i/ISA Debugger 1040, INTMON 1041, MDEBUG 1042, Soft-ICE 1050, Turbo Debugger 1052, Miscellaneous 1054

**CHAPTER 58 Remote Control Software** **1059**

Carbon Copy Plus 1059, COMMUTE 1059, pcANYWHERE 1060, Miscellaneous 1064

**CHAPTER 59 Viruses and Anti-Viral Tools** **1067**

Viruses 1067, Anti-Viral Tools 1074

**CHAPTER 60 ZIPKEY** **1081**

**CHAPTER 61 PC Tools** **1085**

**CHAPTER 62 Norton Utilities** **1101**

**CHAPTER 63 Miscellaneous Quarterdeck Programs** **1105**

**CHAPTER 64 FAX Software** **1109**

BitFax Scheduler 1109, FaxBIOS 1110, MTEZ XpressFax 1122, RainbowFAX 1122, ZyXEL ZFAX 1124, Miscellaneous 1125

**CHAPTER 65 Communicating Applications Specification** **1127**

**CHAPTER 66 Miscellaneous APIs** **1139**

Installation Checks only 1139, APCAL 1145, ASCII 1146, AutoCAD 1146, CU Writer 1148, FSBBS 1149, GOLD 1151, Jot-It! 1151, Kingswood TSR Windows 1152, METZTSR 1153, PCL-838 1154, PC Thuis Shell and Vanderaart Text Windows 1155, PCW Weather Card 1160, RAID 1161, RighTime 1161, Screen Thief 1162, SETDRVER 1163, SitBack 1164, StackMan 1165, Trusted Access 1166, VEDIT VSWAP 1167, WHOA! 1167, XVIEW 1168, Miscellaneous 1169

**Index** **1179**

## CHAPTER 1

# *Introduction*

Welcome to *PC Interrupts*, second edition. You are holding what is probably the largest quick-reference guide ever published; together with its smaller companion book *Network Interrupts* (to be available shortly after this book), it forms the most comprehensive reference to IBM PC-related interrupt calls. The split into two companion books is our solution to the dilemma we faced when starting the second edition: how to cram more than 1800 pages of information into the 1200-page limit imposed by the publisher's equipment (a two-volume set was vetoed by the marketing department) without omitting important calls. Although some material has been greatly abbreviated, we were able to include all non-networking calls of which we were aware at the time of writing into this book.

## Why This Book Exists

The IBM PC and compatibles form the largest portion of the computer market, and likely will continue to do so into the next century. The MS-DOS operating system for IBM PCs is the most popular ever developed, and the brisk sales of the recently released MS-DOS 6.0 proves that it will remain so for some time. Because the architecture of the IBM PC allows programs to provide services to other programs via software interrupts, and MS-DOS provides a mechanism for retaining programs in memory in order to continue providing such services, hundreds of software vendors have extended the basic set of services in often incompatible ways. As more programs provide additional interrupt calls, the probability of conflict between different programs increases.

Over the years, literally dozens of books have been written which include reference sections detailing the ROM BIOS services built into IBM PCs, and the additional services provided by MS-DOS. Some also cover the Expanded Memory Specification (EMS) services, but few go beyond that. Thus, programmers have been forced to accumulate information about additional interrupt services piecemeal — one document covering one specification, another a second, and an electronic bulletin-board message a third. With such a collection of sources, it can be quite difficult to determine that two programs are incompatible because they expect different services from the same interrupt call.

This book is a comprehensive collection of the varied calls which have been implemented by various programs over the years. It includes all calls for some 50 major application programming interfaces and dozens of resident utilities (including shareware programs and undocumented interfaces), as well as the usual coverage of BIOS and MS-DOS services. Because of the number of different calls covered (over 4000), there is no room for detailed discussions or sample code.

PC Interrupts is designed to be a reference, not a tutorial. It will form a companion to any texts covering MSDOS or the IBM PC which you may already own. In addition to its main purpose as a programming reference, you may also find it valuable as a tool to help you track down undesirable interactions between various programs running on the same PC.

The material presupposes some familiarity with programming using software interrupts, although there is no need to know a particular programming language. Software interrupts may be invoked from most high-level languages for the IBM PC as well as assembler, and most books on those languages cover interrupts at least briefly.

## Interrupts and the '86 CPU Family

While this book is intended as a reference rather than as a tutorial, so many different partial listings of interrupt usage have been published, each using its own vocabulary and assumptions, that it is necessary to take a few pages up front to present an overview of the subject. This section briefly describes the basic interrupt concept as implemented in the Intel 80x86 family of processor.

The Intel 80x86 family of processors all provide the capability of servicing a maximum of 256 distinct interrupt actions, by means of a table of 256 four-byte interrupt vectors. This table occupies the first 1024 bytes of the RAM address space. Each vector consists of a 32-bit address, in standard Segment:Offset format, of the routine that will be called automatically in response to the corresponding interrupt.

The interrupt request that triggers the calling of such a routine can be generated in any of three ways: it can be internally generated by the processor chip itself, it can be created by an external interrupt request signal (called INTR by Intel), or it can be the result of a software interrupt (INT) instruction.

The original members of this processor family, the 8086 and 8088, only dedicated the first five interrupt vectors (INT 00h through INT 04h) to serving internally generated interrupts, although Intel clearly warned all users of the chips that the remaining 27 of the first 32 interrupts were reserved for future use. IBM chose to ignore that warning in the design of the PC, and assigned INT 05h to the print-screen action, while using the upper 24 of the 32 reserved interrupts for communication with hardware devices and BIOS. Of these 24 vectors, the first eight were assigned to deal with external device interrupts (INT 08h through 0Fh, for IRQ0 through IRQ7 respectively). The remaining 16 provided, via software INT requests, standardized interfaces to the BIOS routines.

When the next version of the processor, the 80186, appeared, problems ensued. Trade journals reported that the conflict between IBM's use of the reserved locations, and Intel's subsequent assignment of them, were the reason that no IBM machine used the 80186, but this was never publicly confirmed. However the following chip, the 80286, was more restrained in its assignment of interrupt actions. Subsequent designs have followed its lead; all of the first 32 locations now have internal actions assigned for at least some versions of the processor family, but all remain compatible with MS-DOS usage.

So far as the processor's actions are concerned, there is almost no difference between an interrupt caused by one of the sources and another caused by any other. The only significant difference is that external interrupts automatically disable response to additional external interrupt requests, while software interrupts do not.

## Some Words of Caution

Much of the information in this book is undocumented, and has therefore been determined by tracing calls and by trial-and-error experiments. Such undocumented information may be inaccurate, incomplete, highly version-dependent, or have any mix of these attributes. You should take care when attempting to use any calls marked as "internal " or "undocumented ", or containing a large amount of italicized text (signifying its questionable or incomplete nature). Further, you should use documented equivalents whenever possible, even if the undocumented call is simpler or faster.

Version information has been included in many function descriptions; note, however, that this information represents only the version limits that we (or our contributors) were able to verify.

Especially in the case of third-party software, any function may well have been present but undetected in versions earlier than those listed. And open-ended items such as "Version 2.1+ " (meaning "all versions equal to or newer than 2.1 ") may have a hidden upper limit imposed by changes not reported to us before this information went to press.

## Sample Entry

To illustrate some of the aspects of the entries in this book which may not be self-explanatory, we present a mythical function followed by explanations of various parts of the entry.

---

**INT FFh Function 0Fh** *XYZZY models 17 and 23 only*
***GET GONKULATOR SETTINGS*** *FROB.SYS*

**Purpose:** Determine the current or default options for the gonkulator.

**Registers at call:**
AH = 0Fh
AL = subfunction
  01h get default settings
  02h get current settings
    ES:DI -> settings table (Table 99-2)

**Return Registers:**
CF clear if successful
  CX = number of times settings have been modified since startup
  ES:DI buffer filled
CF set on error
  AX = error code (02h,FFh) (see Table 99-1)

**Details:** The link farm directory is the location for the write-only memory, among other things.

**Conflicts:** DIDDLE.SYS (Chapter 96)

**See also:** Function 0Eh, DIDDLE.SYS INT FEh Function 02h (Chapter 96), FOO-NET INT FEh Function FEh (NI)

**99-1 Format of settings table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | heartbeat frequency in Hertz |
| 02h | BYTE | feeper duration |
| 03h | BYTE | feeper pitch |
| 04h | DWORD | microfortnights between pings |
| 08h | 64 BYTEs | link farm directory name |

### *Explanations*

Following the heading which indicates the interrupt number, function number, and subfunction number (if applicable) along with the function name, restrictions to particular systems or software, and other limitations such as processor mode, each entry has a number of fields. Both Registers at call and Return Registers may have multiple mutually-exclusive cases, which are indicated by indentation. Thus, for this example, the call modifies CX and the user's buffer if the carry flag is clear on return, but modifies AX instead if the carry flag is set.

The cross-references for *Conflicts* and *See also* indicate which chapter contains the referenced call, unless it is in the current chapter or in *Network Interrupts* (marked as NI). Thus, the example has two references to Chapter 96, one to the current chapter, and one to *Network Interrupts.* Conflicts specifies calls that use the same interrupt and function; you would look in Chapter 99 for interrupt FFh Function 0Fh (or, in some cases, simply interrupt FFh) in the section for DIDDLE.SYS if the chapter is divided into sections. For *See also*, the interrupt number is omitted if it is the same as the interrupt for the current entry; thus, you would look up interrupt FFh, function 0Eh in the current chapter.

In the various sections and in tables, information which is known to be incomplete or of questionable accuracy is italicized. This generally occurs only for undocumented calls.

## About The Authors

**Ralf Brown**

Ralf Brown is an independent software developer and author who recently received a Ph.D. in Computer Science from Carnegie Mellon University. He has delved into the innards of MSDOS and IBM PC compatibles since early 1984, and is well-known in the on-line community for maintaining the MSDOS Interrupt List and authoring a number of useful free programs. He coauthored *Undocumented DOS* (Addison-Wesley 1993).

**Jim Kyle**

Jim Kyle has been a professional writer since 1948, and has published more than a dozen books and hundreds of magazine articles, including coauthoring *Undocumented DOS* (Addison-Wesley 1993). He has been disassembling operating systems as a hobby since 1970 or so, on mainframes and minicomputers as well as microcomputers. He has been Primary Forum Administrator of the Computer Language forum on CompuServe since 1985. Jim is currently Director, Research Projects, and a co-founder of Automation Resources, Inc., in Oklahoma City, OK.

## Acknowledgments

The information in this book has been adapted from a large freely-available electronic listing, known as the MSDOS Interrupt List, maintained by Ralf Brown and updated several times per year. The Interrupt List is available in many places, including but not limited to:

- CompuServe (IBM Programming Forum [GO IBMPRO],Download Library 6)
- Internet (oak.oakland.edu and mirror sites such as wuarchive.wustl.edu, where it resides in /mirrors/msdos/info)
- BITNET (any of a number of mailservers, such as LISTSERV@COL2 = PIECS in the United States or TRICKLE@DS0RUS1I in Germany)
- Fidonet (bulletin boards belonging to the Prorammer's Distribution Network, as well as others)
- other bulletin boards world-wide.

The list is distributed under the filenames INTERrrA through INTERrrD, where rr is the release number (36 for the August, 1993 version to which this edition corresponds). Unlike PC Interrupts, the electronic listing is in purely numerical order and does not contain an index.

Many people have contributed information to the Interrupt List over the years, resulting in a much larger listing than would otherwise have been the case. We gratefully acknowledge the contributions of the following people:

A. Padgett Peterson, A. Peter Blicher, Aaron Emigh, Aki Korhonen, Alex Fedorov, Alexi Lookin, Andrea Omodeo, Andrei Tsyganenko, Andrew Rossmann, Andrew Schulman, Andrew Torda, Andy Hakim, Ari Huttunen, Arne Schapers, Barry Burke, Ben Castricum, Ben Myers, Bent Lynggaard, Bernd Schemmer, Bert Baker, Bill Frolik, Bill White, Bob Aitchison, Bob Fehrenbach, Bob Green, Bob Jack, Boris M. Mostovoy, Brad Davis, Brad Templeton, Brett Warthen, Brian Leeming, Brian Long, Brian McGuinness, Bruce Gingery, Carl Schelin, Carlos Antunes, Carlos M. de Sousa, Carlos Rioja, Chris Blum, Chris Dunford, Chris Hall, Christian Franke, Christopher J. Ambler, Ciriaco Garcia de Celis, Clarence A. Dold, Claude Marche, Cornel Kaufmann, Dan Bodoh, Dan Crocker, Dan Davison, Dan Fandrich, Dan Lanciani, Dan Prather, Darryl Gregorash, Dave Andrews, Dave Guggisberg, Dave Kirsch, David Dyck, David Fink, David G. Thomas, David H. Bennett, David Herron, David Maxey, David Woolley, Dennis Grinberg, Diomidis Spinellis, Dmitry Ablov, Dmitry Stefankov, Dr. David Bailey, Drake Koefoed, Duncan Murdoch, Eberhard Mattes, Ed Nather, Ed Palmer, Edwin Cleton, Edwin Floyd, Erik Liljencrantz, Everett C. Johnson, Everett Kaser, Finn Thoegerson, Francis Turner, Francis Turner, Frank Behrens, Frank Gladu, Frank Klemm, Frank van der Linden, Frank Vorstenbosch, Frank-Christian Kruegel, Fridrik Skulason, Fyodor Evseev, G. Adam Stanislav, Gary E. Miller, Ge van Geldorp, Geoff Chappell, George L. Fulk, George Smith, George W. Pogue, Gianni Albóre, Gino Lucrezi, Glen Kriekenbeck, Gordon Edwards, Grant Echols, Greg Pasquariello, Guenther Thiele, Harald Langhammer, Harry Sumar, Heath Ian Hunnicutt, Helmut Waelder, Herwig Feichtinger, Holger Veit, Howard Johnson, Ian Oliver, Igor Sysoev, Inbar Raz, Ivan Martinez, J. B. Gill, J. Weaver Jr., Jack Alvrus, Jack Ridgway, Jacob Rieper, James Berry, James Birdsall, James P. Kiely, James West, Jan Ceuleers, Jan-Pascal van Best, Jan-Pieter Cornet, Janet Jack, Janos Haide, Jean-Francois Larvoire, Jeff Pipkins, Jens Bleuel, Jens Vollmar, Jeremy Laidman, Jeroen Pluimers, Jerzy Tarasiuk, Jim Kyle, Jim Peterson, Jiri Kuchta, Joe Morris, Joe Souza, Johan Zwiekhorst, John B. Thiel, John Cooper, John DesRosiers, John Fa'atuai, John Howells, John Jurewicz, John Lefor, John Murphy, John Navas, John P. Vias, John Ruschmeyer, John Spinks, John Villalovos, John Werner, John.Brennen, Jonathan Story, Jose R. Rodriguez Santia, Joseph G. Souza, Joseph Gil, Ken Medellin, Kevin Gillett, Kevin Gillett, Klaus Fischer, Klaus Hartnegg, Konstantin Kisurin, Krzysztof Halasa, KUCHTA@dcse.fee.vutbr.cs, Kyle Rogers, Leonard Erickson, Les Moskowitz, Lewis Paper, Lutz Schroer, Madis Kaal, Manfred Young, Marc Scholtis, Marco Lumachi, Marcus Groeber, Mark Aitchison, Mark Davis, Mark Livingstone, Mark Scase, Mark Seiffert, Mark T. Garlanger, Mark T. Vitt, Martin Shipley, Martin Wilde, Martin Winkler, Marty Leisner, Maxime Dallaire, Michael A. Moran, Michael A. Shiels, Michael Bootz, Michael D. Lawler, Michael D. Shride, Michael Dickson, Michael Ho, Michael Hung, Michael L. Kaufman, Michael S. Stratoti, Michel Mathieu, Mikael Rydberg, Mike Baszczak, Mike Morearty, Mike Weaver, Mitch Davis, Morten Welinder, Naota Kimura, Nelluri Reddy, Nemrod Kedem, Nigel Bree, noesis@ucscb.UCSC.EDU, Norbert Juffa, Norbert Sommer, Norman Walsh, Otto J. Makela, Otto-Michael Braun, Patrick Ibbetson, Paul Ratcliffe, Paul van Keep, Paul Vojta, Paul Williamson, Pavel Shtemenko, Per-Eric Larsson, Pete Fales, Pete Peterson, Peter Holzmann, Peter Sawatzki, Peter Singer, Rainer Schuetze, Ralf Brown, Ralph Heredia, Rehn-Lieh Lin, Rich Goldschmidt, Richard A. Plinston, Richard D. Morris, Richard Hargrove, Richard Marks, Richard Talon, Rick Wagner, Rickard Faith, Riku Meskanen, Risto Lankinen, Rob Luursema, Robert Chafer, Robert Goldsmith, Robert Seals, Robin Walker, Roeland Jansen, Roger Bowler, Roman Rutman, Ronald Lokker, Ross M. Greenberg, Ross Wentworth, RS Tse, Russ Nelson, Scott C. Pedigo, Sean Lin, Serge Krupa, Serge Pachkovsky, Serge Pachkovsky, Sergey Danilov, Sergio Fogel, Shalom Krischer, Simon Phipps, Skip Gilbrech, Skule Johansen, Stan Brown, Stefan Gustafsson, Stephan Wolf, Stephen Bean, Steve Bromwich, Steve Grant, Steve Halko, Stuart R. Kemp, Sylvan Butler, Tamura Jones, Thomas Dwyer, Thomas Hundt, Thomas Lukka, Tim Farley, Timo Salmi, Todor Todorov, Tom Rawson, Torbj|rn Lindgren, Urs Zurbuchen, Victor Poon, Vincent Broman, Vsevolod M. Shabad, W.F. Schroeder, Warner Young, Wes Cowley, Wilbert Van Leijen, William Peavy, Wim Osterholt, Wolfgang Lierz, Xavier Cabelle, Ying-Kuen K. Hwang, Yousuf Khan

Special thanks also to the following companies for making API information and other materials available at no charge:

Alloy Computer Products, Inc
Banyan Systems, Inc.
General Software
Helix Software Company, Inc.
Intel Literature
International Computers
Microsoft Corporation
Net-Source, Inc.
Nanosoft, Inc.
NetManage, Inc.
Novell, Inc.
Phar Lap Software, Inc.
Phoenix Technologies, Ltd.
Quarterdeck Office Systems
Bernd Schemmer
Tiara Computer Systems, Inc.
V Communications, Inc.
Video Electronics Standards Association
Webcorp

## CHAPTER 2

# *Organization and Master Interrupt List*

The brief listings in Tables 2-1 through 2-4 show which chapters describe functions for each of the 256 possible interrupts. Those referenced for more than one chapter or program probably have conflicts under some conditions.

Two areas in particular have a high probability of conflicts. The first is the multiplex interrupt, INT 2Fh (Table 2-2), and the second is the user interrupts, INT 60h to INT 67h.

Interrupt 2Fh is shared by many programs, with the value of AH on call specifying the program which is to handle the call. As more programs use the multiplex interrupt to provide an interface to resident code, the probability of conflicting multiplex numbers (the value in AH) increases. While there is a standardized method for determining whether a particular multiplex number is in use (though not all programs follow the convention), there is no standardized approach for determining whether a specific program is using it. Thus, a program may be fooled into thinking that it is already installed if a different program happens to be using the same multiplex number, or worse yet, make calls which are handled by a program other than the intended one, possibly with disastrous results.

We consider a two-level scanning approach such as that used by Quarterdeck's RPCI (Chapter 63) the minimum for avoiding conflicts between programs on INT 2Fh, but recommend using the Alternate Multiplex Interrupt Specification described in Chapter 50 instead of INT 2Fh. AMIS-compliant TSRs automatically avoid conflicts (including hotkey conflicts) and provide standardized calls for removal and popup while imposing an overhead of less than 100 bytes in the resident code; they may also be removed in any order.

One problem caused by the lack of a written specification for INT 2Fh usage is that not all programs use the convention of returning AL=FFh when called with AL=00h, and many that do support the convention also modify other registers—including segment registers or BP—even without the presence of "magic " values on entry. Therefore, any program which searches a range of multiplex numbers must be prepared for all registers except SS:SP to be destroyed by an attempted installation check.

The user interrupts 60h to 67h do not have a convention for an installation-check call, and thus require a different approach. However, the user interrupts are rarely chained, so a check whether the vector is 0000h:0000h suffices to determine if a particular interrupt is available. To determine whether the vector is in use by the program making the installation check, the usual approach is to place a signature immediately prior to or immediately following the interrupt handler address, since there are no safe calls which can be made—the vector may not even point at code! Table 2-3 lists programs we have identified as using this group of interrupts, but like Table 2-2, it is almost certain to be incomplete.

A final area of conflict needs to be mentioned here, as it is the reason so few programs use interrupts 86h through F0h. These vectors are used by the BASIC interpreter in the ROM of genuine IBM machines to allow extensions and tracing. The BASIC.COM and BASICA.COM extensions to the ROM Cassette BASIC set these vectors to handlers inside themselves at startup, but never restore them. Thus, any resident programs which have hooked any of these interrupts will lose them if the user runs an interpreted BASIC program. Worse, the vectors will be pointing at whatever code or data happens to have been loaded into those locations since BASIC terminated, virtually guaranteeing a system crash. GWBASIC (the version for those without an IBM ROM containing BASIC), on the other hand, only uses interrupts EFh and F0h, and restores those on exit.

**2-1 Overview of Interrupt Usage**

| *INT* | *Usage* |
|---|---|
| 00h | Divide Error (Chapter 3), Zenith ROM Debugger (Chapter 57) |
| 01h | Single-Step/Debugging Exceptions (Chapter 3) |
| 02h | Non-Maskable Interrupt (Chapter 3) |
| 03h | Breakpoint (Chapter 3); Columbia ROM Debugger, Soft-ICE (Chapter 57) |
| 04h | Overflow Detected (Chapter 3) |
| 05h | Bounds Exceeded (Chapter 3), Print Screen (Chapter 4), PSPS (Chapter 15) |
| 06h | Invalid Opcode (Chapter 3), HP 95LX (Chapter 7) |
| 07h | Coprocessor Not Present (Chapter 3) |
| 08h | Double Fault, IRQ0 (Chapter 3) |
| 09h | Coprocessor Protection Violation, IRQ1 (Chapter 3) |
| 0Ah | Invalid TSS, IRQ2 (Chapter 3) |
| 0Bh | Segment Not Present, IRQ3 (Chapter 3); HP 95LX (Chapter 7) |
| 0Ch | Stack Violation, IRQ4 (Chapter 3); IBM System 36/38 Worstation Emulation (NI) |
| 0Dh | General Protection Violation, IRQ5 (Chapter 3); HP 95LX (Chapter 7) |
| 0Eh | Page Fault, IRQ6 (Chapter 3); HP 95LX (Chapter 7) |
| 0Fh | IRQ7 (Chapter 3), HP 95LX (Chapter 7) |
| 10h | Coprocessor Error (Chapter 3); AX PC Video (Chapter 6); Corona/Cordata BIOS (Chapter 7); BIOS Video, Chips&Technologies Extended BIOS, Compaq Extensions, Everex Extended Video BIOS, HP Vectra Extended BIOS, EGA Register Interface Library, BIOS Window Extension, Compaq ADAPT.COM, M10_SCR, Hercules GRAFIX, SCROLOCK, UltraVision, Netroom SCRNCLK, MSHERC, Realtek RTVGA, Direct Graphics Interface Standard, SCREENR, Show Partner F/X, VUIMAGE (Chapter 8); SuperVGA Video, Cirrus Logic BIOS, VESA SuperVGA BIOS, SOLLEX SuperVGA (Chapter 9); FASTBUFF (Chapter 13); FRIEZE (Chapter 15); Tinytalk Personal (Chapter 23); GO32 DOS extender (Chapter 37); TopView, DESQview Video (Chapter 44); Carbon Copy Plus (Chapter 58); CU Writer (Chapter 66); 3270PC (NI), Alloy MW386 (NI) |
| 11h | Alignment Check (Chapter 3), Get Equipment List (Chapter 4), SDLP (Chapter 11), BNU FOSSIL (Chapter 12), Blank-It screen blanker (Chapter 14), Back&Forth (Chapter 49), Borland C++ IDE (Chapter 54), RainbowFAX (Chapter 64) |
| 12h | Get Memory Size (Chapter 4), KEYBUI (Chapter 13), Back&Forth (Chapter 49), Borland C++ IDE (Chapter 54), PARKER (Chapter 66) |
| 13h | Disk I/O, ESDI Hard Disk, Priam EDVR.SYS, SWBIOS, Western Digital SuperBIOS (Chapter 10); Adaptec AHA-154x BIOS, Future Domain SCSI (Chapter 11); FAST! disk cache, HyperDisk, IBMCACHE.SYS, NOW! disk cache, QUICKCACHE II, Super PC-Kwik (Chapter 17); ScanBoot, VSAFE/VWATCH (Chapter 59); Beame&Whiteside BWLPD (NI) |
| 14h | Video FOSSIL (Chapter 8); Serial I/O, COMM-DRV, COURIERS.COM, Digiboard DigiCHANNEL, IBM/Yale EBIOS, FOSSIL, MBBIOS, MX5 Extended |

| | |
|---|---|
| | FOSSIL, TSRCOMM (Chapter 12); Keyboard FOSSIL (Chapter 13); ASAP (Chapter 23); MultiDOS Plus IODRV (Chapter 45); PC-MOS/386 $serial.sys (Chapter 49); 3com BAPI (NI), ARTICOM (NI), Alloy MW386 (NI), Beame&Whiteside BWCOM14 (NI), TelAPI (NI), Interconnections Inc. TES (NI) |
| 15h | ABIOS, Cassette I/O, OS hooks, BIOS System functions, keyboard intercept, joystick, extended memory access, PS/2 DMA control, PS/2 Device Descriptor Tables (Chapter 4); EISA System ROM (Chapter 5); Compaq extensions, HP 95LX, Phoenix 386 BIOS (Chapter 7); Amstrad PC1512 (Chapters 7 and 21); ESDI disk formatting (Chapter 10); PRINT (Chapter 15); Advanced Power Management specification (Chapter 19); PS/2 Pointing Device interface (Chapter 21); Far East MS-DOS (Chapter 24); Rational Systems DOS/16M and DOS/4G (Chapter 35); Tandon Memory Mapper (Chapter 42); DESQview, TopView, DESQview/X (Chapter 44); MultiDOS Plus (Chapter 45); VMiX (Chapter 46); Omniview multitasker (Chapter 49); Microsoft TSR Specification (Chapter 51) |
| 16h | Keyboard (Chapter 4); AX PC keyboard (Chapter 6); Compaq extensions, HP HIL Vectras (Chapter 7); EGAPAL and VGAPAL (Chapter 8); FAKEY.COM, M16_KBD.COM, STACKEY.COM, TEXTCAP (Chapter 13); Netroom CACHECLK (Chapter 17); Netroom (Chapter 41); CPTASK (Chapter 49); Swap Utilities, TSRBONES, TurboPower TSRs (Chapter 51); Paint Tools PTxxx.COM (Chapter 54); pcANYWHERE (Chapter 58); PC Tools CPSCHED/DESKTOP/PCShell/PCRUN (Chapter 61); Norton Guides (Chapter 62); APCAL, AT.COM, CALCULATOR, FastJuice, JORJ, MAKEY.COM, PC Magazine PUSHDIR, SCOUT, TMED, Borland Turbo Lightning, WATCH.COM, Microsoft Word (Chapter 66); PC Tools DRIVEMAP (NI), Shamrock Software EMAIL (NI) |
| 17h | AX PC Printer (Chapter 6); FLASHUP.COM, SPEEDSCR.COM (Chapter 8); BIOS Printer Services, Emulaser ELTSR, INSET, NorthNet JetStream, LPTx, PC-MOS/386 print spooler, PC Paint Plus, PC Magazine PCSpool, T2PS (Chapter 15); Shamrock Software NET.24 (NI), Alloy networks (NI) |
| 18h | Diskless Boot hook (Chapter 4), NEC PC-9800 (Chapter 6) |
| 19h | Bootstrap Loader (Chapter 4) |
| 1Ah | BIOS Time Services (Chapter 4); Intel PCI BIOS, PCMCIA Socket Services (Chapter 5); NEC PC-9800 (Chapter 6); AT&T 6300 (Chapter 7); Disk Spool II, Emulaser ELSPL (Chapter 15); Microsoft Real-Time Compression Interface (Chapter 16); PCjr sound, SND, Tandy digital sound (Chapter 22); DATEFIX, RighTime, Word Perfect interface (Chapter 66); SNAP (NI) |
| 1Bh | Control-Break handler (Chapter 4) |
| 1Ch | System Timer Tick (Chapter 4) |
| 1Dh | Video Parameter Tables (Chapter 4) |
| 1Eh | Diskette Parameters (Chapter 10) |
| 1Fh | 8x8 Graphics Font (Chapter 8) |
| 20h | MS-DOS Program Termination (Chapter 24), Minix (Chapter 49), COMTROL Hostess (Chapter 57) |
| 21h | PGS1600.DEV, Compaq AG1024.SYS (Chapter 8); xDISK (Chapter 10); ASPI, ASPITAPE.SYS, ST-01 SCSI.SYS (Chapter 11); CED, DOSED, WCED (Chapter 13); CUBIT, DBLSPACE.BIN, DIET, Diet Disk, DUBLDISK.SYS, NewSpace, PCMANAGE/DCOMPRES, Stacker (Chapter 16); COMBI-disk, Super PC-Kwik, SMARTDRV.SYS (Chapter 17); FARTBELL (Chapter 22); MS-DOS (Chapter 24); European MS-DOS (Chapter 25); DR-DOS/Novell DOS, Concurrent DOS, DR Multiuser DOS, OS/2 Family API, OS/2 Virtual DOS Machine (Chapter 26); CD-ROM device driver, SHARE (Chapter 27); Intel Code Builder, Phar Lap 386\|DOS- |

| | |
|---|---|
| | Extender, FlashTek X-32VM (Chapter 33); OS/286 and OS/386 (Chapter 34); DOS/4GW (Chapter 35); AI Architects, GO32 (Chapter 37); QEMM-386 (Chapter 39); 386MAX (Chapter 40); The Last Byte, Memory Managers, Microsoft EMM386.EXE (Chapter 42); DESQview (Chapter 44); CTask, DoubleDOS, Headroom, PC-Mix, PC-MOS/386, Software Carousel (Chapter 49); COMTROL Hostess, StopPrg, Turbo Debugger hardware breakpoints (Chapter 57); pcANYWHERE IV/LAN (Chapter 58); Virus, F-Prot, FLU_SHOT+, VDEFEND/VSAFE/VWATCH (Chapter 59); DOS-UP.SYS, HOOKROM.SYS (Chapter 63); SoftLogic Data Guardian, ELRES, IBM Genie, PCW Weather Card, SCROLLit, TAME, Trusted Access, WATCH.COM, WILDUNIX (Chapter 66); DOS network support (NI), Alloy NTNX (NI), Banyan VINES (NI), BW-TCP (NI), CBIS PowerLAN (NI), Attachmate Extra (NI), FTPSOFT.DOS (NI), LAN Manager (NI), LANstep (NI), LANtastic (NI), Named Pipes interface (NI), NDIS (NI), Novell NetWare (NI), Sun PC-NFS (NI), PC/TCP (NI), 10NET (NI), WORKGRP.SYS (NI), IBM System 36/38 Workstation Emulation (NI), Topware Network Operating System (NI) |
| 22h | Program Termination Address (Chapter 24), COMTROL Hostess (Chapter 57) |
| 23h | Control-C/Control-Break Handler (Chapter 24), COMTROL Hostess (Chapter 57) |
| 24h | Critical Error Handler (Chapter 24) |
| 25h | Stacker (Chapter 16), PC-CACHE.SYS (Chapter 17), Absolute Disk Read (Chapter 24) |
| 26h | Absolute Disk Write (Chapter 24), COMTROL Hostess (Chapter 57) |
| 27h | Terminate and Stay Resident (Chapter 24), COMTROL Hostess (Chapter 57) |
| 28h | DOS Idle (Chapter 24) |
| 29h | Fast Console Output (Chapter 24) |
| 2Ah | PRINT (Chapter 15), Network (NI), NetBIOS (NI), LANtastic (NI), PC Network (NI), IBM 3270 Emulation Program (NI), Novell NetWare Lite (NI), PC/TCP PREDIR (NI) |
| 2Bh | reserved by MS-DOS (Chapter 24) |
| 2Ch | reserved by MS-DOS (Chapter 24), STARLITE architecture (Chapter 26), Cloaking API (Chapter 32) |
| 2Dh | reserved by MS-DOS (Chapter 24), Alternate Multiplex Interrupt Specification (Chapter 50) and AMIS TSRs: Burnout Plus (Chapter 14), dLite (Chapter 16), RATSR (Chapter 58), RAMLIGHT (Chapter 66), Screen Thief (Chapter 66) |
| 2Eh | COMMAND.COM back door, 4DOS SHELL2E.COM (Chapter 28); Windows NT native API (Chapter 43) |
| 2Fh | Multiplex Interrupt — see Table 2-2 |
| 30h | DOS CP/M entry point (Chapter 24) |
| 31h | DOS CP/M entry point (Chapter 24), DOS Protected-Mode Interface (Chapter 30), Netroom3 DPMI.EXE (Chapter 41), Virus (Chapter 59) |
| 32h | reserved by MS-DOS (Chapter 24), Virus (Chapter 59) |
| 33h | Mouse, Smooth Mouse Driver (Chapter 21); Switch-It (Chapter 49) |
| 34h-37h | Floating Point emulation (Chapter 54) |
| 38h | PC-MOS/386 (Chapter 49), Floating Point emulation (Chapter 54) |
| 39h-3Eh | Floating Point emulation (Chapter 54) |
| 3Fh | Overlay manager, MS Dynamic Link Library manager (Chapter 54) |
| 40h | Acorn BBC Master 512, Zenith Z100 (Chapter 7); Floppy Diskette services (Chapter 10) |
| 41h | Acorn BBC Master 512, Zenith Z100 (Chapter 7); Hard Disk 0 parameter table (Chapter 10); MS Windows debugging kernel (Chapter 57) |

| | |
|---|---|
| 42h | Acorn BBC Master 512, Zenith Z100 (Chapter 7); EGA-relocated INT 10h services (Chapter 8); Western Digital SuperBIOS (Chapter 10) |
| 43h | Acorn BBC Master 512, Zenith Z100 (Chapter 7); EGA+ character table (Chapter 8) |
| 44h | Acorn BBC Master 512, Zenith Z100 (Chapter 7); PCjr BIOS character font (Chapter 8); Virus (Chapter 59); Novell NetWare (NI), IBM 3270-PC HLLAPI (NI) |
| 45h | Acorn BBC Master 512, Zenith Z100 (Chapter 7) |
| 46h | Acorn BBC Master 512, Zenith Z100 (Chapter 7); Hard Disk 1 parameter table (Chapter 10) |
| 47h | Acorn BBC Master 512, Zenith Z100 (Chapter 7); Western Digital SuperBIOS (Chapter 10); SQL Base (Chatper 54) |
| 48h | Acorn BBC Master 512, Zenith Z100 (Chapter 7); Western Digital SuperBIOS (Chapter 10); PCjr keyboard translation (Chapter 13); Compaq UILIB.EXE (Chapter 54); Watstar PC Network (NI) |
| 49h | Acorn BBC Master 512, Zenith Z100 (Chapter 7); Texas Instruments PC video (Chapter 8); PCjr scan-code translation table (Chapter 13); MAGic (Chapter 23); Watstar PC Network (NI) |
| 4Ah | User Alarm Handler (Chapter 4); Acorn BBC Master 512, Zenith Z100 (Chapter 7) |
| 4Bh | Acorn BBC Master 512, Zenith Z100 (Chapter 7); Common Access Method SCSI interface, IBM SCSI interface (Chapter 11); Virtual DMA Specification (Chapter 18) |
| 4Ch | Acorn BBC Master 512, Zenith Z100 (Chapter 7) |
| 4Dh | Acorn BBC Master 512 (Chapter 7) |
| 4Eh | Zenith Z100 (Chapter 7), TI Professional PC disk (Chapter 10) |
| 4Fh | Zenith Z100 (Chapter 7), Common Access Method SCSI interface (Chapter 11) |
| 50h | OS/2 relocated IRQ0 (Chapter 26), DESQview relocated IRQ0 (Chapter 44), Vanderaart Text Windows (Chapter 66), PC Thuis Shell (Chapter 66), IBM 3278 emulation relocated IRQ0 (NI), TIL Xpert AIM (NI) |
| 51h | OS/2 relocated IRQ1 (Chapter 26), DESQview relocated IRQ1 (Chapter 44), IBM 3278 emulation relocated IRQ1 (NI) |
| 52h | OS/2 relocated IRQ2 (Chapter 26), DESQview relocated IRQ2 (Chapter 44), IBM 3278 emulation relocated IRQ2 (NI) |
| 53h | OS/2 relocated IRQ3 (Chapter 26), DESQview relocated IRQ3 (Chapter 44), IBM 3278 emulation relocated IRQ3 (NI), WEB (NI) |
| 54h | OS/2 relocated IRQ4 (Chapter 26), DESQview relocated IRQ4 (Chapter 44), IBM 3278 emulation relocated IRQ4 (NI) |
| 55h | OS/2 relocated IRQ5 (Chapter 26), DESQview relocated IRQ5 (Chapter 44), IBM 3278 emulation relocated IRQ5 (NI) |
| 56h | OS/2 relocated IRQ6 (Chapter 26), DESQview relocated IRQ6 (Chapter 44), IBM 3278 emulation relocated IRQ6 (NI) |
| 57h | OS/2 relocated IRQ7 (Chapter 26), DESQview relocated IRQ7 (Chapter 44), IBM 3278 emulation relocated IRQ7 (NI) |
| 58h | DESQview relocated IRQ8 (Chapter 44), DoubleDOS relocated IRQ0 (Chapter 49) |
| 59h | GSS Computer Graphics Interface (Chapter 8), DESQview relocated IRQ9 (Chapter 44), DoubleDOS relocated IRQ1 (Chapter 49) |
| 5Ah | DESQview relocated IRQ10 (Chapter 44), DoubleDOS relocated IRQ2 (Chapter 49), Cluster Adapter (NI) |
| 5Bh | DESQview relocated IRQ11 (Chapter 44), DoubleDOS relocated IRQ3 (Chapter 49), SitBack (Chapter 66), Cluster Adapter (NI), AT&T Starlan Extended NetBIOS (NI), Microsoft Network Transport Layer (NI), Alloy NTNX (NI), ISOLAN Multi Protocol Software (NI) |

| | |
|---|---|
| 5Ch | DESQview relocated IRQ12 (Chapter 44), DoubleDOS relocated IRQ4 (Chapter 49), NetBIOS (NI), TOPS (NI), ATALK.SYS (NI), IBM 802.2 interface (NI), $25 LAN (NI) |
| 5Dh | DESQview relocated IRQ13 (Chapter 44), DoubleDOS relocated IRQ5 (Chapter 49) |
| 5Eh | DESQview relocated IRQ14 (Chapter 44), DoubleDOS relocated IRQ6 (Chapter 49) |
| 5Fh | HP 95LX Graphics Primitives (Chapter 20), DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49) |
| 60h-67h | reserved for user interrupts — see Table 2-3 |
| 68h | APPC/PC (NI), Sangoma CCPOP 3270 (NI) |
| 69h | Zenith AT BIOS (Chapter 7), ISR.COM (Chapter 57), DECnet DOS (NI), 10NET SYSSVC.COM (NI) |
| 6Ah | OPTHELP (Chapter 66), DECnet DOS (NI), Super-TCP kernel (NI) |
| 6Bh | Virus (Chapter 59), Tandy Schoolmate Plus (Chapter 66), DECnet DOS (NI), Novell NASI/NACS (NI), TelAPI (NI), NCSI (NI) |
| 6Ch | Convertible System Resume Vector (Chapter 4), DOS 3.2 Realtime Clock update (Chapter 24), DECnet DOS (NI) |
| 6Dh | VGA internal, ATI VGA Wonder, Trident SVGA (Chapter 9); DECnet DOS (NI) |
| 6Eh | DECnet DOS (NI) |
| 6Fh | HP HIL Vectras (Chapter 7), MS Windows (Chapter 43), Novell NetWare 3270 API (NI), 10NET (NI) |
| 70h | IRQ8 Real-Time Clock (Chapter 3), Virus (Chapter 59) |
| 71h-76h | IRQ9 to IRQ14 (Chapter 3) |
| 77h | IRQ15, Compaq Power Conservation (Chapter 3) |
| 78h | ADP-60 IDE controller (Chapter 10); TARGA.DEV (Chapter 11); DBOS DOS Extender, GO32 (Chapter 37); AutoCAD Device Interface (Chapter 66) |
| 79h | ADP-60 IDE adapter (Chapter 10); AVATAR.SYS (Chapter 13); DBOS DOS Extender, GO32 (Chapter 37); AutoCAD Device Interface (Chapter 66) |
| 7Ah | PRINDIR (Chapter 15), GO32 (Chapter 37), AutoCAD Device Interface (Chapter 66), Canon IX-30F Image Scanner (Chapter 66), Novell NetWare (NI), IBM 3270 Workstation Program (NI), Topware Network Operating System (NI), X.PC Packet (NI) |
| 7Bh | GO32 (Chapter 37), Btrieve (Chapter 54), AutoCAD Device Interface (Chapter 66), Eicon Access (NI) |
| 7Ch | PRINDIR (Chapter 15), GO32 (Chapter 37), REXX88PC (Chapter 66), SK-UPPS (NI) |
| 7Dh | SCSILink (Chapter 11), GO32 (Chapter 37), HyperPAD (Chapter 66), YTERM (Chapter 66) |
| 7Eh | GO32 (Chapter 37), XLOAD (Chapter 41), DIP, Ltd. ROM Library (Chapter 66), YTERM (Chapter 66) |
| 7Fh | see Table 2-4 |
| 80h | SoundBlaster SBFM driver (Chapter 22), reserved for BASIC (Chapter 54), Q-PRO (Chapter 66), QPC Software PKTINT (NI) |
| 81h | reserved for BASIC (Chapter 54), IBM Token Ring Adapter (NI) |
| 82h | reserved for BASIC (Chapter 54), IBM Token Ring Adapter (NI) |
| 83h-85h | reserved for BASIC (Chapter 54) |
| 86h | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54), NetBIOS (NI) |
| 87h | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54), Virus (Chapter 59) |
| 88h | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54) |
| 89h | IBM ROM BASIC (Chapter 54) |

| | |
|---|---|
| 8Ah | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54) |
| 8Bh | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54), Virus (Chapter 59) |
| 8Ch | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54) |
| 8Dh-8Fh | IBM ROM BASIC (Chapter 54) |
| 90h | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54) |
| 91h | IBM ROM BASIC (Chapter 54), IBM Token Ring Adapter (NI) |
| 92h | IBM ROM BASIC (Chapter 54), Sangoma X.25 interface (NI), DaVinci eMail Dispatcher (NI) |
| 93h | IBM ROM BASIC (Chapter 54), IBM Token Ring Adapter (NI) |
| 94h | PCM driver (Chapter 22), Media Vision PCM.COM (Chapter 22), IBM ROM BASIC (Chapter 54) |
| 95h | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54) |
| 96h-9Bh | IBM ROM BASIC (Chapter 54) |
| 9Ch-9Fh | IBM ROM BASIC (Chapter 54), Virus (Chapter 59) |
| A0h | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54) |
| A1h-A3h | IBM ROM BASIC (Chapter 54) |
| A4h | IBM ROM BASIC (Chapter 54), Right Hand Man (Chapter 66) |
| A5h-B2h | IBM ROM BASIC (Chapter 54) |
| B3h | IBM ROM BASIC (Chapter 54), ZIPKEY (Chapter 60) |
| B4h | IBM ROM BASIC (Chapter 54), StackMan (Chapter 66) |
| B5h | Netroom NETSWAP4 (Chapter 41), IBM ROM BASIC (Chapter 54), StackMan (Chapter 66) |
| B6h-BDh | IBM ROM BASIC (Chapter 54) |
| BEh | DESQview/X (Chapter 44), IBM ROM BASIC (Chapter 54) |
| BFh-C5h | IBM ROM BASIC (Chapter 54) |
| C6h-CFh | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54) |
| D0h | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54), NJFRERAM (Chapter 66) |
| D1h-D3h | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54) |
| D4h | PC-MOS/386 (Chapter 49), APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54) |
| D5h-D7h | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54) |
| D8h-DBh | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54), Screen Thief (Chapter 66) |
| DCh | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54), Screen Thief (Chapter 66), PC/370 (Chapter 66) |
| DDh-DEh | APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54), Screen Thief (Chapter 66) |
| DFh | Victor 9000 (Chapter 7), APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54), Screen Thief (Chapter 66) |
| E0h | DR Multiuser DOS (Chapter 48), APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54), Virus (Chapter 59) |
| E1h-E2h | IBM ROM BASIC (Chapter 54), PC Cluster (NI) |
| E3h | IBM ROM BASIC (Chapter 54) |
| E4h | IBM ROM BASIC (Chapter 54), Logitech Modula (Chapter 54) |
| E5h-EBh | IBM ROM BASIC (Chapter 54) |
| ECh | Exact (Chapter 54), IBM ROM BASIC (Chapter 54), Alloy NTNX (NI) |
| EDh-EEh | IBM ROM BASIC (Chapter 54) |
| EFh | BASIC (Chapter 54), GEM (Chapter 54) |
| F0h | BASIC, BASICA (Chapter 54) |

| | |
|---|---|
| F1h | reserved for user interrupt: AQUEDUCT/PIPELINE (Chapter 12), SPEECH.COM (Chapter 22), Virus (Chapter 59) |
| F2h | reserved for user interrupt: ICCTSR (Chapter 8), SPEECH.COM (Chapter 22) |
| F3h | reserved for user interrupt: ICCTSR (Chapter 8), SoundBlaster (Chapter 22) |
| F4h | reserved for user interrupt: DoubleDOS (Chapter 49) |
| F5h | reserved for user interrupt: DoubleDOS (Chapter 49) |
| F6h | reserved for user interrupt: DoubleDOS (Chapter 49) |
| F7h | reserved for user interrupt: DoubleDOS (Chapter 49), FSBBS (Chapter 66) |
| F8h | reserved for user interrupt: DoubleDOS (Chapter 49) |
| F9h | reserved for user interrupt: DoubleDOS (Chapter 49) |
| FAh | reserved for user interrupt: DoubleDOS (Chapter 49) |
| FBh | reserved for user interrupt: DoubleDOS (Chapter 49) |
| FCh | reserved for user interrupt: DoubleDOS (Chapter 49) |
| FDh | reserved for user interrupt: TFPCX (Chapter 12), DoubleDOS (Chapter 49) |
| FEh | destroyed by return from protected mode (Chapter 4), DoubleDOS (Chapter 49), Turbo Debugger (Chapter 57) |
| FFh | destroyed by return from protected mode (Chapter 4), Z100 Warm Boot (Chapter 7), QEMM-386 (Chapter 39), Virus (Chapter 59) |

**2-2 INT 2Fh Multiplex Number Usage**

| *Program* | *Chapter* | *Low* | *Default* | *High* | *Selection* |
|---|---|---|---|---|---|
| Ross Wentworth POPUP library | 51 | 00h | N/A | FFh | compile-time |
| CiriSOFT University of Valladolid TSRs | 51 | 00h | 00h | FFh | automatic |
| PRINT.COM | 15 | . | 00h | . | fixed |
| PSPRINT | 15 | . | 00h | . | fixed |
| PRINT.COM | 15 | . | 01h | . | fixed |
| PC LAN Program redirector | NI | . | 02h | . | fixed |
| Critical Error Message Expansion | 27 | . | 05h | . | fixed |
| ASSIGN | 27 | . | 06h | . | fixed |
| DRIVER.SYS support | 24 | . | 08h | . | fixed |
| SuperStor Pro SSTORDRV.SYS | 16 | . | 10h | . | fixed |
| DR-DOS SHARE support | 26 | . | 10h | . | fixed |
| SHARE | 27 | . | 10h | . | fixed |
| MSCDEX | 27 | . | 11h | . | fixed |
| Network Redirector interface | NI | . | 11h | . | fixed |
| LAN Manager | NI | . | 11h | . | fixed |
| MS-DOS Network Redirector support | 24, NI | . | 12h | . | fixed |
| DR-DOS video memory space control | 26 | . | 12h | . | fixed |
| DOS 3.2+ disk handler addresses | 24 | . | 13h | . | fixed |
| MS-NET | NI | . | 13h | . | fixed |
| European MS-DOS POPUP | 25 | . | 14h | . | fixed |
| DR-DOS NLSFUNC | 26 | . | 14h | . | fixed |
| MS-DOS NLSFUNC | 27 | . | 14h | . | fixed |
| PC-DOS 4.00 GRAPHICS.COM | 27 | . | 15h | . | fixed |
| CD-ROM | NI | . | 15h | . | fixed |
| MS Windows DOS-app interface | 43 | . | 16h | . | fixed |
| DOS Protected-Mode Interface (DPMI) | 30 | . | 16h | . | fixed |
| MS Windows WINOLDAP API | 43 | . | 17h | . | fixed |
| *Program* | *Chapter* | *Low* | *Default* | *High* | *Selection* |
| MS-Manager | 66 | | 18h | . | fixed |

| | | | | | |
|---|---|---|---|---|---|
| DOS 4.x SHELLB.COM | 27 | . | 19h | . | fixed |
| ANSI.SYS | 27 | . | 1Ah | . | fixed |
| ANSIPLUS.SYS | 8 | . | 1A h | . | fixed |
| AVATAR.SYS | 8 | . | 1Ah | . | fixed |
| AVATAR Serial Dispatcher | 12 | . | 1Ah | . | fixed |
| XMA2EMS.SYS | 27 | . | 1B h | . | fixed |
| DR-DOS GRAFTABL | 26 | . | 23h | . | fixed |
| DR-DOS TaskMAX | 47 | . | 27h | . | fixed |
| Gammafax DOS Dispatcher | 64 | . | 2Ah | . | fixed |
| Kingswood TSR interface | 51 | . | 39h | . | fixed |
| Kingswood TSR Windows | 66 | . | 39h | . | fixed |
| OS/2 compatibility box | 26 | . | 40h | . | fixed |
| Phar Lap 286\|DOS-Extender Lite | 33 | . | 40h | . | fixed |
| Windows callouts | 43 | . | 40h | . | fixed |
| LAN Manager MINIPOP/NETPOPUP | NI | . | 41h | . | fixed |
| LAN Manager messenger service | NI | . | 42h | . | fixed |
| DOS Protected Mode Services (DPMS) | 31 | . | 43h | . | fixed |
| Extended Memory Specification (XMS) | 38 | . | 43h | . | fixed |
| HIMEM.SYS | 42 | . | 43h | . | fixed |
| DOS Extender Support | 37 | . | 44h | . | fixed |
| Microsoft Profiler | 56 | . | 45h | . | fixed |
| DOS 5+ kernel | 24 | . | 46h | . | fixed |
| Windows/286 DOS Extender | 37 | . | 46h | . | fixed |
| F-PROT v1.x | 59 | . | 46h | . | fixed |
| MS Windows v3.x installation check | 43 | . | 46h | . | fixed |
| *unknown* | 66 | . | 47h | . | fixed |
| DOSKEY | 13 | . | 48h | . | fixed |
| PCED | 13 | . | 48h | . | fixed |
| *unknown* | 66 | . | 49h | . | fixed |
| DoubleSpace | 16 | . | 4Ah | . | fixed |
| Microsoft Real-Time Compression I'face | 16 | . | 4Ah | . | fixed |
| SMARTDRV | 17 | . | 4Ah | . | fixed |
| Floppy-Disk Change notification | 24 | . | 4Ah | . | fixed |
| HMA allocation | 24 | . | 4Ah | . | fixed |
| Diskless Boot memory sizing | 27 | . | 4Ah | . | fixed |
| PROTMAN support | 27 | . | 4Ah | . | fixed |
| DOSSHELL Task Switching API | 49 | . | 4Ah | . | fixed |
| MS Windows WIN.COM | 43 | . | 4Bh | . | fixed |
| DOS Task Switcher interface | 49 | . | 4Bh | . | fixed |
| LAN Manager NETWKSTA.EXE | NI | . | 4Bh | . | fixed |
| Advanced Power Management | 19 | . | 4Ch | . | fixed |
| Kana Kanji Converter | 66 | . | 4Dh | . | fixed |
| SilverNET | NI | . | 4Eh | . | fixed |
| POWER.EXE | 19 | . | 53h | . | fixed |
| POWER.EXE | 19 | . | 54h | . | fixed |
| TesSeRact RAM-resident interface | 51 | . | 54h | . | fixed |
| COMMAND.COM interface | 28 | . | 55h | . | fixed |

| *Program* | *Chapter* | *Low* | *Default* | *High* | *Selection* |
|---|---|---|---|---|---|
| INTERLNK | NI | . | 56h | . | fixed |
| Iomega Corp. | 10 | . | 57h | . | fixed |
| PC Tools VDEFEND/VSAFE/ VWATCH/DATAMON | 59 | . | 62h | . | fixed |
| SCRNSAV2.COM | 14 | . | 64h | . | fixed |
| License Service API | NI | . | 70h | . | fixed |
| SRDISK | 10 | . | 72h | . | fixed |
| Novell NetWare IPX/SPX/etc. | NI | . | 7Ah | . | fixed |
| LAN HiJack | NI | . | 7Ah | . | fixed |
| Jim Harper's CD-ROM redirector | 11 | . | 7Fh | . | fixed |
| FaxBIOS | 64 | . | 80h | . | fixed |
| ASCII | 66 | . | 80h | . | fixed |
| EASY-NET | NI | . | 80h | . | fixed |
| Nanosoft TurboNET server | NI | . | 80h | . | fixed |
| Nanosoft TurboNET redirector | NI | . | 81h | . | fixed |
| RESPLAY | 22 | . | 82h | . | fixed |
| Nanosoft CAPDOS | 57 | . | 82h | . | fixed |
| Codeview for Windows | 57 | . | 86h | . | fixed |
| WHOA! | 66 | . | 89h | . | fixed |
| RAID | 66 | . | 90h | . | fixed |
| Courier LAN E-Mail | NI | . | 92h | . | fixed |
| PC Tools DRIVEMAP | NI | . | 92h | . | fixed |
| InnerMission | 14 | . | 93h | . | fixed |
| Workgroup Connection MICRO.EXE | NI | . | 94h | . | fixed |
| Courier LAN E-Mail OPERATOR.EXE | NI | . | 9Ch | . | fixed |
| INTMON | 57 | . | 9Eh | . | fixed |
| INTCFG | 57 | . | 9Fh | . | fixed |
| Ergo DOS Extenders | 34 | . | A1h | . | fixed |
| Biologic HRAMDEV.SYS | 42 | . | A1h | . | fixed |
| Futurus Team | 66 | . | A4h | . | fixed |
| METZTSR.COM | 66 | 80h | A9h | FFh | switch |
| VIDCLOCK | 66 | . | AAh | . | fixed |
| Btrieve Multi-User | 54 | . | ABh | . | fixed |
| DOS 4.01+ GRAPHICS.COM | 27 | . | ACh | . | fixed |
| DR-DOS KEYB | 26 | . | ADh | . | fixed |
| MS-DOS DISPLAY.SYS | 27 | . | ADh | . | fixed |
| MS-DOS KEYB | 27 | . | ADh | . | fixed |
| COMMAND.COM installable commands | 28 | . | AEh | . | fixed |
| MS-DOS GRAFTABL | 27 | . | B0h | . | fixed |
| IBM PC3270 Emulation Program | NI | . | B4h | . | fixed |
| APPEND | 27 | . | B7h | . | fixed |
| Network | NI | . | B8h | . | fixed |
| 10NET installation check | NI | . | B8h | . | fixed |
| DOS LAN Requester | NI | . | B8h | . | fixed |
| PC Network | NI | . | B9h | . | fixed |
| Media Vision MVSOUND.SYS | 22 | . | BCh | . | fixed |
| EGA.SYS | 43 | . | BCh | . | fixed |

| *Program* | *Chapter* | *Low* | *Default* | *High* | *Selection* |
|---|---|---|---|---|---|
| REDVIEW | 66 | . | BEh | . | fixed |
| PC LAN Program REDIRIFS | NI | . | BFh | . | fixed |
| WANG_ER.COM | 10 | C0h | C0h | FFh | automatic |
| ASPIHOOK.SYS | 11 | C0h | C0 | FFh | automatic |
| After Dark AD-DOS | 14 | C0h | C0h | FFh | automatic |
| Explosiv | 14 | C0h | C0h | C9h | config |
| VGAsave | 14 | C0h | C0h | FFh | automatic |
| Cove Software QMR | 21 | C0h | C0h | FFh | automatic |
| MTEZ XpressFax | 64 | C0h | C0h | FFh | automatic |
| FN32 | 66 | . | C0h | . | fixed |
| Novell ODI Link Support Layer (LSL) | NI | C0h | C0h | FFh | automatic |
| Smart Prompt (SMARTPMT) | 17 | . | C6h | . | fixed |
| ThunderByte | 59 | . | C9h | . | fixed |
| StackMan | 66 | . | C9h | . | fixed |
| TBSCANX | 59 | . | CAh | . | fixed |
| FAXPLUS FAXTSR | 64 | . | CAh | . | fixed |
| THELP v3.0 | 66 | . | CAh | . | fixed |
| Comwave Microfax Specification | 64 | . | CBh | . | fixed |
| MTEZ XpressFax CASMGR | 64 | . | CBh | . | fixed |
| Communicating Appliations Specification | 65 | . | CBh | . | fixed |
| Intel SatisFAXtion CASMGR | 65 | . | CBh | . | fixed |
| PC-Kwik programs | 17 | . | CCh | . | fixed |
| LaserPort interface | 15 | . | CDh | . | fixed |
| SWELL.EXE | 49 | . | CDh | . | fixed |
| Intel Image Processing Interface | 55 | . | CDh | . | fixed |
| TEMPLEXX | 13 | . | CFh | . | fixed |
| ZWmous | 21 | . | D0h | . | fixed |
| MDEBUG display driver | 57 | 01h | D0h | FEh | config, API |
| Lotus CD/Networker | NI | . | D0h | . | fixed |
| MDEBUG command driver | 57 | 02h | D1h | FFh | config ,API |
| Quarterdeck RPCI | 63 | C0h | D2h | FFh | automatic |
| PCL-838.EXE | 66 | . | D2h | . | fixed |
| LapLink Quick Connect | 12 | . | D3h | . | fixed |
| TeleReplica | 58 | . | D3h | . | fixed |
| 4DOS (4DOS.COM, KSTACK.COM) | 28 | . | D4h | . | fixed |
| HEART.COM | 57 | . | D6h | . | fixed |
| VEDIT VSWAP | 66 | . | D6h | . | fixed |
| Banyan VINES | NI | . | D7h | . | fixed |
| Novell NetWare Lite | NI | . | D8h | . | fixed |
| TRAP.COM | 57 | . | DAh | . | fixed |
| ZyXEL ZFAX | 64 | . | DAh | . | fixed |
| Beame&Whiteside BWSNMP | NI | . | DAh | . | fixed |
| GOLD.COM | 66 | . | DCh | . | fixed |
| xDISK | 10 | . | DDh | . | fixed |
| CappaCom programs | 51 | . | DDh | . | fixed |
| DESQview External Device Interface | 44 | C0h | DEh | FFh | automatic |
| HyperWare programs (HyperDisk, etc.) | 17 | C0h | DFh | FFh | automatic |
| StuffIt | 13 | . | E0h | . | fixed |

| *Program* | *Chapter* | *Low* | *Default* | *High* | *Selection* |
|---|---|---|---|---|---|
| SETDRVER.COM | 66 | . | E0h | . | fixed |
| Phantom of the Keyboard II | 13 | . | E1h | . | fixed |
| ANARKEY | 13 | C0h | E3h | FFh | switch |
| NDOS | 28 | . | E4h | . | fixed |
| Phar Lap DOS Extenders | 33 | . | EDh | . | fixed |
| GRIDLOC | 66 | . | EEh | . | fixed |
| XVIEW | 66 | . | EEh | . | fixed |
| WEB | NI | . | EEh | . | fixed |
| 4MAP | 13 | . | F0h | . | fixed |
| MIN-MEM | 66 | . | F1h | . | fixed |
| DOS Extender installation check | 37 | . | F1h | . | fixed |
| WINX | 43 | . | F2h | . | fixed |
| AUTOPARK | 66 | . | F7h | . | fixed |
| SuperStor Pro 2XON | 16 | . | F8h | . | fixed |
| SoundBlaster speech driver | 22 | . | FBh | . | fixed |
| AutoBraille | 23 | . | FBh | . | fixed |
| Borland DOS Extenders (DPMILOAD, etc.) | 36 | . | FBh | . | fixed |
| Borland TDX | 57 | . | FBh | . | fixed |
| Jot-It! | 66 | . | FBh | . | fixed |
| Norton Utilities | 62 | . | FEh | . | fixed |
| Sun PC-NFS | NI | . | FEh | . | fixed |
| Topware Network Operating System | NI | . | FFh | . | fixed |
| BMB Compuscience Canada Utilities | 51 | 00h | FFh | FFh | automatic |

### 2-3 User Interrupt Usage

| *Program* | *Chapter* | *60h* | *61h* | *62h* | *63h* | *64h* | *65h* | *66h* | *67h* | *Selection* |
|---|---|---|---|---|---|---|---|---|---|---|
| SYS_PROF | 56 | D | . | . | . | . | . | . | . | fixed |
| Zero Bug virus | 59 | D | . | . | . | . | . | . | . | fixed |
| 3com | NI | D | . | . | . | . | . | . | . | fixed |
| Excelan LAN Workplace for DOS 3.5 | NI | D | . | . | . | . | . | . | . | fixed |
| AccessDOS | 23 | D | a | a | a | a | a | a | . | automatic |
| Nabbit | 66 | D | a | a | a | a | a | a | . | automatic |
| Banyan VINES | NI | D | a | a | a | a | a | a | . | automatic |
| Tangram Arbiter | NI | D | a | a | a | a | a | a | . | config |
| INTRSPY/CMDSPY | 57 | D | a | a | a | a | a | a | a | automatic |
| Buffit | 66 | D | a | a | a | a | a | a | a | automatic |
| PC-IPC | 66 | D | a | a | a | a | a | a | a | switch |
| PC/370 | 66 | D | a | a | a | a | a | a | a | patch |
| FTP Packet Driver Specification | NI | D | a | a | a | a | a | a | a | automatic |
| MDEBUG | 57 | D | a | a | a | a | a | a | a | config/autodete |
| Atari Portfolio | 7 | D | D | . | . | . | . | . | . | fixed |
| HP 95LX System Manager | 20 | D | D | . | . | . | . | . | . | fixed |
| JPI TopSPEED Modula-2 | 54 | D | D | . | . | . | . | . | . | fixed |
| Adaptec and OMTI controller drive 0 | 10 | D | D | D | D | . | . | . | . | fixed |
| SEMTEX/Screen Trasher virus | 59 | . | D | . | . | . | . | . | . | fixed |
| OPTIMA/ET-3000 Zoom TSR | 8 | a | D | a | a | a | a | a | a | config/autodete |

| *Program* | *Chapter* | *60h* | *61h* | *62h* | *63h* | *64h* | *65h* | *66h* | *67h* | *Selection* |
|---|---|---|---|---|---|---|---|---|---|---|
| TCPOpen kernel | NI | a | D | a | a | a | a | a | a | config |
| FTP Software PC/TCP kernel | NI | a | D | a | a | a | a | a | a | config/autodetect |
| Sangoma CCIP (CCPOP 3270) | NI | . | D | . | . | . | . | . | D | fixed |
| HP 95LX calculator | 20 | . | . | D | . | . | . | . | . | fixed |
| Cswitch | 49 | . | . | D | . | . | . | . | . | fixed |
| FastGraph/Light FGDRIVER | 52 | . | . | D | . | . | . | . | . | fixed |
| MS SQL Server/Sybase DBLIBRARY | 54 | . | . | D | . | . | . | . | . | fixed |
| PC Tools COMMUTE | 58 | . | . | D | . | . | . | . | . | fixed |
| BW-TCP hardware driver (ETHDEV.SYS) | NI | a | a | D | a | a | a | a | . | config |
| 4+Power Floppy Controller | 10 | . | . | . | D | . | . | . | . | fixed |
| HP 100LX memory mapping | 20 | . | . | . | D | . | . | . | . | fixed |
| Kofax KF9X00 image manipulation card | 66 | . | . | . | D | . | . | . | . | fixed |
| DESQview/X Socket API | NI | . | . | . | D | . | . | . | . | fixed |
| BW-TCP TCP/IP stack | NI | . | a | a | D | a | a | a | a | config |
| Oracle SQL Protected Mode Executive | 54 | . | . | . | . | D | D | . | . | fixed |
| Extended Batch Language | 66 | . | . | . | . | . | D | . | . | fixed |
| Novell NetWare Low-Level API | NI | . | . | . | . | . | D | . | . | fixed |
| BW-NFS BWRPC | NI | . | . | a | a | D | a | a | a | config |
| Data General DG10 MicroECLIPSE | 66 | . | . | . | . | . | D | D | D | fixed |
| Adaptec controllers drive 1 | 10 | . | . | . | . | D | D | D | D | fixed |
| Pdisk hard drive information | 10 | . | . | . | . | D | D | D | D | fixed |
| EZRECV | 12 | . | . | . | . | . | . | D | . | fixed |
| Ad Lib SOUND.COM | 22 | . | . | . | . | . | . | D | . | fixed |
| Media Vision FM.COM | 22 | . | . | . | . | . | . | D | . | fixed |
| SD.COM | 66 | . | . | . | . | . | . | D | . | fixed |
| FTP Software NDIS-Packet Driver adapter | NI | . | . | . | . | . | D | . | . | fixed |
| Nanosoft TurboNET | NI | . | . | . | . | . | . | . | D | fixed |
| Microsoft Windows VITD.386 | 43 | . | . | . | . | . | . | . | D | fixed |
| Bitfax Scheduler | 64 | . | . | . | . | . | . | . | D | fixed |
| IBMSND driver | 22 | . | . | . | . | . | . | . | D | fixed |
| IBM Digitized Sound Package MIDI driver | 22 | . | . | . | . | . | . | . | D | fixed |
| PenDOS | 66 | . | . | . | . | . | . | . | D | fixed |
| MicroHelp Stay-Res | 51 | . | . | . | . | . | . | . | D | fixed |
| Virtual Control Program Interface |  | . | 29 |  |  | . | . | . | D | fixed |
| Expanded Memory Specification (EMS) | 38 | . | . | . | . | . | . | . | D | fixed |
| Quarterdeck QEMM installation check | 39 | . | . | . | . | . | . | . | D | fixed |
| Qualitas 386MAX MEMLIMIT | 40 | . | . | . | . | . | . | . | D | fixed |
| Netroom RM386 | 41 | . | . | . | . | . | . | . | D | fixed |
| MICEMM | 41 | . | . | . | . | . | . | . | D | fixed |

| *Program* | *Chapter* | *60h* | *61h* | *62h* | *63h* | *64h* | *65h* | *66h* | *67h* | *Selection* |
|---|---|---|---|---|---|---|---|---|---|---|
| Compaq CEMM | 42 | . | . | . | . | . | . | . | D | fixed |
| Microsoft EMM386.EXE | 42 | . | . | . | . | . | . | . | D | fixed |
| Nanosoft MD386 | 42 | . | . | . | . | . | . | . | D | fixed |
| Quadtel QMAPS | 42 | . | . | . | . | . | . | . | D | fixed |
| PC-NET | NI | . | . | . | . | . | . | . | D | fixed |
| Newkey | 13 | a | a | a | a | a | a | a | D | automatic |
| CUCKOO | 66 | a | a | a | a | a | a | a | D | automatic |

**Key: D** = default interrupt, **a** = alternate, **.** = not usable by program

## 2-4 INT 7Fh Usage

| *Chapter* | *Program* |
|---|---|
| 8 | Halo88 |
| 8 | HDILOAD 8514/A controller interface |
| 8 | IBM 8516 Touch Screen Device Driver |
| 8 | IBM XGA |
| 12 | TIGA Communications Driver |
| 37 | GO32 DOS Extender |
| 49 | MultiLink Advanced |
| 54 | Btrieve Multi-User |
| 66 | Canon IXHND2 Scanner Interface |
| 66 | YTERM |
| NI | Alloy 386/MultiWare (MW386), NTNX, and ANSK |
| NI | Convergent Technologies ClusterShare CTOS Access Vector |
| NI | HLLAPI |
| NI | NetWare 2.x File Server |
| NI | Telebit ACS Serial I/O |

## CHAPTER 3

# *Hardware Interrupts*

This chapter describes hardware interrupts, both those generated internally and those which arrive from external devices. Any interrupt signal described in this chapter may occur at any time.

## CPU-generated Interrupts

---

**INT 00h** *CPU-generated*

***DIVIDE ERROR***

**Purpose:** Generated if the divisor of a DIV or IDIV instruction is zero or the quotient overflows the result register; DX and AX will be unchanged.

**Details:** On an 8086/8088, the return address points to the following instruction. On an 80286+, the return address points to the divide instruction. An 8086/8088 will generate this interrupt if the result of a division is 80h (byte) or 8000h (word).

**Conflicts:** Zenith ROM Debugger (Chapter 57)

**See also:** INT 04h

---

**INT 01h** *CPU-generated*

***SINGLE STEP***

**Purpose:** Generated after each instruction if TF (trap flag) is set; TF is cleared on invoking the single-step interrupt handler.

**Details:** Interrupts are prioritized such that external interrupts are invoked after the INT 01h pushes CS:IP/FLAGS and clears TF, but before the first instruction of the handler executes.

Used by debuggers for single-instruction execution tracing, such as MS-DOS DEBUG's T command.

**See also:** INT 03h

---

**INT 01h** *CPU-generated*

***DEBUGGING EXCEPTIONS*** *80386 and higher*

**Purpose:** Generated by the CPU on various occurrences which may be of interest to a debugger program.

**Details:** Events which may trigger the interrupt:

- Instruction address breakpoint fault - will return to execute instruction
- Data address breakpoint trap - will return to following instruction
- General detect fault, debug registers in use
- Task-switch breakpoint trap
- undocumented 386/486 opcode F1h - will return to following instruction.

**See also:** INT 03h

---

**INT 03h** *CPU-generated*

### *BREAKPOINT*

**Purpose:** Generated by the one-byte breakpoint instruction (opcode CCh).
**Details:** Used by debuggers to implement breakpoints, such as MS-DOS DEBUG's G command.

Also used by Turbo Pascal versions 1,2,3 when {$U+} specified.

Return address points to byte following the breakpoint instruction.
**Conflicts:** Columbia ROM Debugger (Chapter 57), Soft-ICE (Chapter 57)
**See also:** INT 01h

---

**INT 04h** *CPU-generated*

### *INTO DETECTED OVERFLOW*

**Purpose:** The INTO instruction will generate this interrupt if OF (Overflow Flag) is set; otherwise, INTO is effectively a NOP.
**Details:** May be used for convenient overflow testing (to prevent errors from propagating) instead of JO or a JNO/JMP combination.
**See also:** INT 00h

---

**INT 05h** *CPU-generated*

### *BOUND RANGE EXCEEDED*

*80186 and higher*

**Purpose:** Generated by BOUND instruction when the value to be tested is less than the indicated lower bound or greater than the indicated upper bound.
**Details:** Returning from this interrupt re-executes the failing BOUND instruction.
**Conflicts:** BIOS PrintScreen (Chapter 4), PSPS (Chapter 15)

---

**INT 06h** *CPU-generated*

### *INVALID OPCODE*

*80286 and higher*

**Purpose:** This interrupt is generated when the CPU attempts to execute an invalid opcode (most protected-mode instructions are considered invalid in real mode) or a BOUND, LDS, LES, or LIDT instruction which specifies a register rather than a memory address.
**Details:** Return address points to beginning of invalid instruction.

With proper programming, this interrupt may be used to emulate instructions which do not exist; many 386 BIOSes emulate the 80286 undocumented LOADALL instruction which was removed from the 80386+.

Generated by the 80386+ when the LOCK prefix is used with instructions other than BTS, BTR, BTC, XCHG, XADD (486), CMPXCHG (486), INC, DEC, NOT, NEG, ADD, ADC, SUB, SBB, AND, OR, or XOR, or any instruction not accessing memory.
**Conflicts:** HP 95LX (Chapter 7)
**See also:** INT 0Ch, INT 0Dh

---

**INT 07h** *CPU-generated*

### *PROCESSOR EXTENSION NOT AVAILABLE*

*80286 and higher*

**Purpose:** This interrupt is automatically called if a coprocessor instruction is encountered when no coprocessor is installed.
**Details:** Can be used to emulate a numeric coprocessor in software.
**See also:** INT 09h

---

**INT 08h** *CPU-generated*

### *DOUBLE EXCEPTION DETECTED*

*80286 and higher*

**Purpose:** Called when multiple exceptions occur on one instruction, or an exception occurs in an exception handler.
**Details:** Called in protected mode if an interrupt above the defined limit of the interrupt vector table occurs.

Return address points at beginning of instruction with errors or the beginning of the instruction which was about to execute when the external interrupt caused the exception.

If an exception occurs in the double fault handler, the CPU goes into SHUTDOWN mode (which circuitry in the PC/AT converts to a reset); this"triple fault" is a faster way of returning to real mode on many 80286 machines than the standard keyboard controller reset.

**Conflicts:** IRQ0 INT 08h (below)

---

**INT 09h** *CPU-generated*

***PROCESSOR EXTENSION PROTECTION ERROR*** *80286 and 80386*

**Purpose:** Called if the coprocessor attempts to access memory outside a segment boundary; it may occur at an arbitrary time after the coprocessor instruction was issued.

**Details:** Until the condition is cleared or the coprocessor is reset, the only coprocessor instruction which may be used is FNINIT; WAIT or other coprocessor instructions will cause a deadlock because the coprocessor is still busy waiting for data.

**Conflicts:** IRQ1 INT 09h (below)

**See also:** INT 07h, Reserved INT 09h below

---

**INT 09h** *internal hardware*

***RESERVED BY Intel (80486)*** *Protected Mode*

**Details:** This exception has been moved to INT 0Dh.

**Conflicts:** IRQ1 INT 09h (below)

**See also:** Processor Extension INT 09h above, INT 0Dh

---

**INT 0Ah** *CPU-generated (80286+)*

***INVALID TASK STATE SEGMENT*** *Protected Mode*

**Purpose:** Automatically called during a task switch if the new TSS specified by the task gate is invalid for any of the following reasons:

- TSS limit is less than 43 (80286) or 103 (80386/80486)
- LDT selector invalid or segment not present
- null SS selector, or SS selector outside LDT/GDT limit
- stack segment is read-only
- stack segment DPL differs from new CPL, or RPL <> CPL
- CS selector is outside LDT/GDT limit or not code
- non-conforming code segment's DPL differs from CPL
- conforming code segment's DPL > CPL
- DS/ES selectors outside LDT/GDT limit or not readable segments.

**Details:** The handler must use a task gate in order to have a valid TSS under which to execute; it must also reset the busy bit in the new TSS.

**Conflicts:** IRQ2 INT 0Ah (below)

**See also:** INT 0Bh

---

**INT 0Bh** *CPU-generated (80286+)*

***SEGMENT NOT PRESENT*** *Protected Mode*

**Purpose:** Generated when loading a segment register if the segment descriptor indicates that the segment is not currently in memory, unless the segment is an LDT (see INT 0Ah) or stack segment (see INT 0Ch) needed by a task switch.

**Details:** May be used to implement virtual memory by loading in segments as they are accessed, clearing the"not present" bit after loading.

**Conflicts:** IRQ3 INT 0Bh (below)

**See also:** INT 0Ah, INT 0Eh

---

**INT 0Ch** *CPU-generated*

***STACK FAULT*** *80286 and higher*

**Purpose:** This interrupt is generated in protected mode on a stack overflow or underflow, or if an inter-level transition or task switch references a stack segment marked"not present"; it is generated in real mode on accessing a word operand at SS:FFFFh.

**Details:** The 80286 will shut down in real mode if SP=1 before a push. On the PC AT and compatibles, external circuitry generates a reset on shutdown.

**Conflicts:** IRQ4 INT 0Ch (below)

**See also:** INT 0Bh, INT 0Dh

---

**INT 0Dh** *CPU-generated*

***GENERAL PROTECTION VIOLATION*** *80286 and higher*

**Purpose:** The CPU generates this interrupt when it detects a protection violation which does not fit under any other category having a separate interrupt.

**Details:** This interrupt is called in real mode when:

- an instruction accesses a memory operand extending beyond offset FFFFh (i.e. WORD at FFFFh or DWORD at FFFDh or higher) in segment CS, DS, ES, FG, or GS
- a PUSH MEM or POP MEM instruction contains an invalid bit encoding in the second byte
- an instruction exceeds the maximum length allowed (10 bytes for 80286, 15 bytes for 80386/80486)
- an instruction wraps from offset FFFFh to offset 0000h.

Called in protected mode on protection violations not covered by INT 06h through INT 0Ch, including:

- segment limit violations
- write to read-only segments
- accesses using null DS or ES selectors
- accesses to segments with privilege greater than CPL
- wrong descriptor type.

This interrupt is called on an 80486 protected-mode floating-point protection fault.

**Conflicts:** IRQ5 INT 0Dh (below)

**See also:** INT 09h, INT 0Ch

---

**INT 0Eh** *CPU-generated*

***PAGE FAULT*** *80386 and higher native mode*

**Purpose:** This interrupt is generated in protected and virtual-86 modes on attempting to access a 4K memory page whose page table entry has the"present" bit cleared.

**Details:** Used to implement virtual memory.

**Conflicts:** IRQ6 INT 0Eh (below)

**See also:** INT 0Bh

---

**INT 11h** *CPU-generated*

***ALIGNMENT CHECK*** *80486 and higher*

**Purpose:** Automatically generated by the CPU when the AC flag is set, the current privilege level is 3, and a misaligned memory access (WORD not on an even address or DWORD not on a multiple of 4) is made.

**Details:** Not all V86 monitors allow the AC flag to be set, such as Turbo Debugger 386.

**Conflicts:** BIOS Equipment List (Chapter 4), SDLP (Chapter 11), BNU FOSSIL (Chapter 12), Blank-It (Chapter 14), BACK&FORTH (Chapter 49), Borland IDE (Chapter 54), RainbowFAX (Chapter 64)

## External Hardware Interrupts

### INT 02h

*external hardware*

### *NON-MASKABLE INTERRUPT*

**Purpose:** Generated by the CPU when the input to the NMI pin is asserted.

**Details:** Return address points to start of interrupted instruction on 80286+.

On the 80286+, further NMIs are disabled until the next IRET instruction, but one additional NMI is remembered by the hardware and will be serviced after the IRET instruction reenables NMIs.

Maskable interrupts may interrupt the NMI handler if interrupts are enabled.

Although the Intel documentation states that this interrupt is typically used for power-failure procedures, Table 3-1 shows that it has many other uses on IBM-compatible machines.

**3-1 Uses for NMI:**

| *Uses* | *Available On* |
|---|---|
| Memory parity error | all except Jr, CONV, and some machines without memory parity Breakout switch on hardware debuggers |
| Coprocessor interrupt | all except Jr and CONV |
| Keyboard interrupt | Jr, CONV |
| I/O channel check | CONV, PS50+ |
| Disk-controller power-on request | CONV |
| System suspend | CONV |
| Real-time clock | CONV |
| System watch-dog timer, time-out interrupt | PS50+ |
| DMA timer time-out interrupt | PS50+ |
| Low battery | HP 95LX |
| Module pulled | HP 95LX. |

### INT 08h

*IRQ0*

### *SYSTEM TIMER*

**Purpose:** Generated 18.2 times per second by channel 0 of the 8254 system timer, this interrupt is used to keep the time-of-day clock updated.

**Details:** Programs which need to be invoked regularly should use INT 1Ch unless they need to reprogram the timer while still keeping the time-of-day clock running at the proper rate.

The default handler is at F000h:FEA5h in IBM PC and 100%-compatible BIOSes.

This interrupt may be masked by setting bit 0 on I/O port 21h.

**Conflicts:** Double-Fault INT 08h (above)

**See also:** IRQ8 INT 70h below, BIOS INT 1Ch (Chapter 4), BIOS INT 4Ah (Chapter 4), DESQview INT 50h (Chapter 44), DoubleDOS INT 58h (Chapter 49), GO32 INT 78h (Chapter 37), Screen Thief INT D8h (Chapter 8)

### INT 09h

*IRQ1*

### *KEYBOARD DATA READY*

**Purpose:** This interrupt is generated when data is received from the keyboard. This is normally a scan code (from either a keypress *or* a key release), but may also be an ACK or NAK of a command on AT-class keyboards.

**Details:** This IRQ may be masked by setting bit 1 on I/O port 21h.

If the BIOS supports an enhanced (101/102-key) keyboard, it calls INT 15h Function 4Fh after reading the scan code from the keyboard and before further processing; all further processing uses the scan code returned from INT 15h Function 4Fh.

The default interrupt handler is at F000h:E987h in 100%-compatible BIOSes.

The interrupt handler performs the actions listed in Table 3-2 for certain special keystrokes.

DRDOS hooks this interrupt to control the cursor shape (underscore/ half block) for overwrite/insert mode.

DR Multiuser DOS hooks this interrupt for cursor shape control and to control whether Ctrl-Alt-Del reboots the current session or the entire system.

**Conflicts:** CPU-generated INT 09h (above)

**See also:** PrintScreen INT 05h (Chapter 4), HP 95LX INT 0Bh (Chapter 7), INT 15h Functions 4Fh and 85h (Chapter 4), BIOS INT 16h (Chapter 4), INT 1Bh (Chapter 4), METZTSR INT 2Fh Function A901h (Chapter 66), DESQview INT 51h (Chapter 44), DoubleDOS INT 59h (Chapter 49), GO32 INT 79h (Chapter 37)

**3-2 Special Keystroke Actions**

| | |
|---|---|
| Ctrl-Break | clear keyboard buffer, place word 0000h in buffer, invoke INT 1Bh, and set flag at 0040h:0071h |
| SysRq | invoke INT 15h Function 85h |
| Ctrl-Numlock | place system in a tight wait loop until next INT 09h |
| Ctrl-Alt-Del | jump to BIOS startup code (either F000h:FFF0h or the destination of the jump at that address) |
| Shift-PrtSc | invoke INT 05h |

**3-3 Values for scan code:**

| | |
|---|---|
| 01h | Esc |
| 02h | 1 ! |
| 03h | 2 @@ |
| 04h | 3 # |
| 05h | 4 $ |
| 06h | 5 % |
| 07h | 6 ^ |
| 08h | 7 & |
| 09h | 8 * |
| 0Ah | 9 ( |
| 0Bh | 0 ) |
| 0Ch | - _ |
| 0Dh | = + |
| 0Eh | Backspace |
| 0Fh | Tab |
| 10h | Q |
| 11h | W |
| 12h | E |
| 13h | R |
| 14h | T |
| 15h | Y |
| 16h | U |
| 17h | I |
| 18h | O |
| 19h | P |
| 1Ah | [ { |
| 1Bh | ] } |
| 1Ch | Enter |
| 1Dh | Ctrl |
| 26h | L |
| 27h | ; : |
| 28h | ’ “ |
| 29h | ‘ |
| 2Ah | Left Shift |
| 2Bh | \ \| |
| 2Ch | Z |
| 2Dh | X |
| 2Eh | C |
| 2Fh | V |
| 30h | B |
| 31h | N |
| 32h | M |
| 33h | , < |
| 34h | . > |
| 35h | / ? |
| 36h | Right Shift |
| 37h | Grey* |
| 38h | Alt |
| 39h | SpaceBar |
| 3Ah | CapsLock |
| 3Bh | F1 |
| 3Ch | F2 |
| 3Dh | F3 |
| 3Eh | F4 |
| 3Fh | F5 |
| 40h | F6 |
| 41h | F7 |
| 42h | F8 |
| 4Bh | LeftArrow |
| 4Ch | Keypad 5 |
| 4Dh | RightArrow |
| 4Eh | Grey+ |
| 4Fh | End |
| 50h | DownArrow |
| 51h | PgDn |
| 52h | Ins |
| 53h | Del |
| 54h | SysRq |
| 56h | left \\| (102-key) |
| 57h | F11 |
| 58h | F12 |
| 5Ah | PA1 |
| 5Bh | F13 |
| 5Ch | F14 |
| 5Dh | F15 |
| 63h | F16 |
| 64h | F17 |
| 65h | F18 |
| 66h | F19 |
| 67h | F20 |
| 68h | F21 |
| 69h | F22 |
| 6Ah | F23 |
| 6Bh | F24 |
| 6Ch | -- |
| 6Dh | EraseEOF |
| 6Fh | Copy/Play |

| | | | | | |
|---|---|---|---|---|---|
| 1Eh | A | 43h | F9 | 72h | CrSel |
| 1Fh | S | 44h | F10 | 74h | ExSel |
| 20h | D | 45h | NumLock | 75h | -- |
| 21h | F | 46h | ScrollLock | 76h | Clear |
| 22h | G | 47h | Home | E0h | prefix code |
| 23h | H | 48h | UpArrow | E1h | prefix code |
| 24h | J | 49h | PgUp | FAh | ACK |
| 25h | K | 4Ah | Grey- | FEh | RESEND |
| | | | | FFh | kbd error/buffer full |

**Note:** Scan codes 56h-E1h are only available on the extended (101/102-key) keyboard and Host Connected (122-key) keyboard; scan codes 5Ah-76h are only available on the 122-key keyboard.

---

### INT 0Ah

*IRQ2*

### *LPT2 (PC), VERTICAL RETRACE INTERRUPT (EGA,VGA)*

**Details:** The TOPS and PCnet adapters use this interrupt request line by default.

DOS 3.2 revectors IRQ2 to a stack-switching routine.

On ATs and above, the physical data line for IRQ2 is labeled IRQ9 and connects to the slave 8259. The BIOS redirects the interrupt for IRQ9 back here.

Under DESQview, only the INT 15h vector and BASIC segment address (the word at 0000h:0510h) may be assumed to be valid for the handler's process.

Many VGA boards do not implement the vertical retrace interrupt, including the IBM VGA Adapter where the traces are either cut or removed.

**Conflicts:** Invalid TSS (above)

**See also:** DESQview INT 52h (Chapter 44), DoubleDOS INT 5Ah (Chapter 49), GO32 INT 7Ah (Chapter 37)

---

### INT 0Ah

*IRQ2*

### *Tandy 1000-series HARD DISK*

**Details:** This interrupt may be masked by setting bit 2 on I/O port 21h.

**Conflicts:** Invalid TSS (above)

**See also:** DESQview INT 52h (Chapter 44), DoubleDOS INT 5Ah (Chapter 49)

---

### INT 0Ah

*IRQ2*

### *ROLAND MPU MIDI INTERFACE*

**Details:** Newer Roland cards and MIDI interfaces by other manufacturers use a jumper-selectable IRQ, but software and hardware generally defaults to IRQ2.

**Conflicts:** Invalid TSS (above)

**See also:** DESQview INT 52h (Chapter 44), DoubleDOS INT 5Ah (Chapter 49), GO32 INT 7Ah (Chapter 37)

---

### INT 0Bh

*IRQ3*

### *SERIAL COMMUNICATIONS (COM2)*

**Purpose:** Automatically asserted by the UART when COM2 needs attention, if the UART has been programmed to generate interrupts.

**Details:** The TOPS and PCnet adapters use this interrupt request line as an alternate.

On PS/2s, COM2 through COM8 share this interrupt; on many PCs, COM4 shares this interrupt.

This interrupt may be masked by setting bit 3 on I/O port 21h.

**Conflicts:** Segment not Present (above), HP 95LX (Chapter 7)

**See also:** INT 0Ch, DESQview INT 53h (Chapter 44), DoubleDOS INT 5Bh (Chapter 49), GO32 INT 7Bh (Chapter 37)

---

**INT 0Ch** *IRQ4*

### *SERIAL COMMUNICATIONS (COM1)*

**Purpose:** Automatically asserted by the UART when COM1 needs attention, if the UART has been programmed to generate interrupts

**BUG:** this vector is modified but not restored by Direct Access v4.0, and may be left dangling by other programs written with the same version of compiled BASIC.

**Details:** On many PC's, COM3 shares this interrupt.

This interrupt may be masked by setting bit 4 on I/O port 21h.

**Conflicts:** Stack Fault (above), IBM System 36/38 Workstation Emulation (NI)

**See also:** INT 0Bh, DESQview INT 54h (Chapter 44), DoubleDOS INT 5Ch (Chapter 49), GO32 INT 7Ch (Chapter 37)

---

**INT 0Dh** *IRQ5*

### *FIXED DISK (PC,XT), LPT2 (AT), reserved (PS/2)*

**Details:** Under DESQview, only the INT 15h vector and BASIC segment address (the word at 0000h:0510h) may be assumed to be valid for the handler's process.

This interrupt may be masked by setting bit 5 on I/O port 21h.

**Conflicts:** General Protection Violation (above), Tandy 1000 and HP 95LX below

**See also:** INT 0Eh, INT 0Fh, DESQview INT 55h (Chapter 44), DoubleDOS INT 5Dh (Chapter 49), GO32 INT 7Dh (Chapter 37)

---

**INT 0Dh** *IRQ5*

### *Tandy 1000 60 Hz RAM REFRESH*

**Purpose:** Used to ensure that the dynamic RAM retains its contents.

**Conflicts:** General Protection Violation (above), IRQ5 above, HP 95LX below

**See also:** DESQview INT 55h (Chapter 44)

---

**INT 0Dh** *HP 95LX*

### *INFRARED INTERRUPT*

**Purpose:** This interrupt is used to perform communications over the infrared data link.

**Conflicts:** General Protection Violation (above), IRQ5 and Tandy 1000 above

**See also:** HP 95LX INT 0Eh below, HP 95LX INT 0Bh (Chapter 7), HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

**INT 0Eh** *IRQ6*

### *DISKETTE CONTROLLER*

**Purpose:** This interrupt is generated by the floppy disk controller on completion of an operation.

**Details:** Default handler is at F000h:EF57h in IBM PC and 100%-compatible BIOSes.

This interrupt may be masked by setting bit 6 on I/O port 21h.

**Conflicts:** Page Fault (above), HP 95LX below

**See also:** INT 0Dh, DESQview INT 56h (Chapter 44), DoubleDOS INT 5Eh (Chapter 49), GO32 INT 7Eh (Chapter 37)

---

**INT 0Eh** *HP 95LX*

### *EXTERNAL CARD INTERRUPT*

**Conflicts:** Page Fault (above), IRQ6 above

**See also:** HP 95LX INT 0Dh above, HP 95LX INT 0Fh below, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

**INT 0Fh** *IRQ7*

### *PARALLEL PRINTER*

**Purpose:** This interrupt is generated by the LPT1 printer adapter when the printer becomes ready.

**Details:** Many printer adapters do not reliably generate this interrupt.

The 8259 interrupt controller generates an interrupt corresponding to IRQ7 when an error condition occurs.

**Conflicts:** HP 95LX below

**See also:** INT 0Dh, DESQview INT 57h (Chapter 44), DoubleDOS INT 5Fh (Chapter 49), GO32 INT 7Fh (Chapter 37)

---

### INT 0Fh

*HP 95LX*

### *REAL-TIME CLOCK*

**Conflicts:** IRQ7 above

**See also:** HP 95LX INT 0Eh above, INT 70h below, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

### INT 10h

*CPU-generated*
*80286 and higher*

### *COPROCESSOR ERROR*

**Purpose:** This interrupt is generated by the CPU when the -ERROR pin is asserted by the coprocessor.

**Details:** AT's and clones usually wire the coprocessor to use IRQ13, but not all get it right.

**Conflicts:** Video (Chapter 8)

**See also:** CPU INT 09h (above), IRQ13 INT 75h below

---

### INT 70h

*IRQ8*

### *CMOS REAL-TIME CLOCK*

**Purpose:** This interrupt is called when the real-time clock chip generates an alarm or periodic interrupt, among others. The periodic interrupt occurs 1024 times per second.

**Notes:** many BIOSes turn off the periodic interrupt in the INT 70h handler unless in an event wait (see INT 15h Functions 83h and 86h).

This interrupt may be masked by setting bit 0 on I/O port A1h.

**Conflicts:** Virus (Chapter 59)

**See also:** INT 08h above, HP 95LX INT 0Fh above, Amstrad INT 15h Function 01h (Chapter 7), BIOS INT 15h Functions 83h and 86h (Chapter 4), BIOS INT 1Ah Function 02h (Chapter 4), DESQview INT 58h (Chapter 44)

---

### INT 71h

*IRQ9*

### *REDIRECTED TO INT 0Ah BY BIOS*

**Details:** This interrupt may be masked by setting bit 1 on I/O port A1h.

The default BIOS handler invokes INT 0Ah for compatibility, since the pin for IRQ2 on the PC expansion bus became the pin for IRQ9 on the AT expansion bus.

Under DESQview, only the INT 15h vector and BASIC segment address (the word at 0000h:0510h) may be assumed to be valid for the handler's process.

**See also:** DESQview INT 59h (Chapter 44)

---

### INT 72h

*IRQ10*

### *RESERVED*

**Details:** This interrupt may be masked by setting bit 2 on I/O port A1h.

**See also:** DESQview INT 5Ah (Chapter 44)

---

### INT 73h

*IRQ11*

### *RESERVED*

**Details:** This interrupt may be masked by setting bit 3 on I/O port A1h.

**See also:** DESQview INT 5Bh (Chapter 44)

## INT 74h *IRQ12*

### *POINTING DEVICE (PS)*

**Details:** This interrupt may be masked by setting bit 4 on I/O port A1h.

Under DESQview, only the INT 15h vector and BASIC segment address (the word at 0000h:0510h) may be assumed to be valid for the handler's process.

**See also:** Mouse INT 33h (Chapter 21), DESQview INT 5Ch (Chapter 44)

## INT 75h *IRQ13*

### *MATH COPROCESSOR EXCEPTION (AT and up)*

Redirected to INT 02h by the BIOS, for compatibility with the PC.

**Details:** This interrupt may be masked by setting bit 5 on I/O port A1h.

Not all clones wire the coprocessor to generate this IRQ; some systems generate an NMI (see INT 02h above) or assert the -ERROR pin on the CPU (see INT 10h above).

Under DESQview, only the INT 15h vector and BASIC segment address (the word at 0000h:0510h) may be assumed to be valid for the handler's process.

**See also:** Coprocessor INT 10h above, DESQview INT 5Dh (Chapter 44)

## INT 76h *IRQ14*

### *HARD DISK CONTROLLER (AT and later)*

**Details:** This interrupt may be masked by setting bit 6 on I/O port A1h.

**See also:** IRQ6 INT 0Eh above, DESQview INT 5Eh (Chapter 44)

## INT 77h *IRQ15*

### *RESERVED (AT,PS)*

**Details:** This interrupt may be masked by setting bit 7 on I/O port A1h.

**Conflicts:** Compaq Power Conservation (Chapter 7)

**See also:** DESQview INT 5Fh (Chapter 44)

## CHAPTER 4

# *ROM BIOS*

This chapter deals primarily with the "standard " ROM BIOS interfaces first established by the original IBM PC and later PCs manufactured by IBM. See the following chapters for vendor-specific extensions.

---

### INT 05h *BIOS*
### *PRINT SCREEN*

**Purpose:** Dump the current text screen to the first printer.
**Details:** Normally invoked by the INT 09h handler when PrtSc key is pressed, but may be invoked directly by applications.

The byte at 0050h:0000h contains the status used by default handler: 00h not active, 01h PrtSc in progress, FFh last PrtSc encountered error.

The default handler is at F000h:FF54h in IBM PC and 100%-compatible BIOSes.
**Conflicts:** Bound Range Exceeded (see Chapter 3)
**See also:** INT 10h Function 12h Subfunction 20h (Chapter 8)

---

### INT 10h *BIOS*
### *VIDEO SERVICES—see Chapter 8*

---

### INT 11h *BIOS*
### *GET EQUIPMENT LIST*

| **Registes at call:** | **Return Registers:** |
|---|---|
| none | AX = BIOS equipment list word (see Table 4-1) |

**Conflicts:** Alignment Check (Chapter 3), SDLP (Chapter 11), BNU FOSSIL (Chapter 12), Blank-It (Chapter 14), BACK&FORTH (Chapter 49), Borland IDE (Chapter 54), RainbowFAX (Chapter 64)
**See also:** INT 12h

**4-1 Bitfields for BIOS equipment list**

| | |
|---|---|
| bit 0 | floppy disk(s) installed (see bits 6-7) |
| bit 1 | 80x87 coprocessor installed |
| bits 2,3 | number of 16K banks of RAM on motherboard (PC only)<br>number of 64K banks of RAM on motherboard (XT only) |
| bit 2 | pointing device installed (PS/2) |
| bit 3 | unused (PS/2) |
| bits 4-5 | initial video mode<br>00 EGA, VGA, or PGA<br>01 40x25 color<br>10 80x25 color<br>11 80x25 monochrome |

| | |
|---|---|
| bits 6-7 | number of floppies installed less 1 (if bit 0 set) |
| bit 8 | DMA support installed (PCjr, some Tandy 1000s, 1400LT) |
| bits 9-11 | number of serial ports installed |
| bit 12 | game port installed |
| bit 13 | serial printer attached (PCjr)<br>internal modem installed (PC/Convertible) |
| bits 14-15 | number of parallel ports installed |

*Compaq and many other 386/486 machines*

| | |
|---|---|
| EAX bit 23 | page tables set so that Weitek coprocessor addressable in real mode |
| EAX bit 24 | Weitek math coprocessor present |

*Compaq Systempro*

| | |
|---|---|
| EAX bit 25 | internal DMA parallel port available |
| EAX bit 26 | IRQ for internal DMA parallel port (if bit 25 set)<br>0 = IRQ5<br>1 = IRQ7 |
| bits 27,28 | parallel port DMA channel<br>00 DMA channel 0<br>01 *DMA channel 0*<br>10 reserved<br>11 DMA channel 3 |

---

### INT 12h

*BIOS*

***GET MEMORY SIZE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| none | AX = kilobytes of contiguous memory starting at absolute address 00000h |

**Details:** This call returns the contents of the word at 0040h:0013h; in PC and XT, this value is set from the switches on the motherboard.

**Conflicts:** PARKER (Chapter 10), KEYBUI (Chapter 13), Back&Forth (Chapter 49), Borland C++ IDE (Chapter 54)

**See also:** INT 11h, DOS Supervisor INT 2Fh Function 4A06h (Chapter 27)

---

### INT 13h

*BIOS*

***DISK SERVICES—see Chapter 10***

---

### INT 14h

*BIOS*

***SERIAL I/O—see Chapter 12***

---

### INT 15h Function 00h

*CASSETTE*
*PC and PCjr only*

***TURN ON TAPE DRIVE'S MOTOR***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 00h | CF set on error<br>AH = 86h no cassette present<br>CF clear if successful |

**Conflicts:** Amstrad mouse (Chapter 21), MultiDOS Plus (Chapter 45), VMiX (Chapter 46)

**See also:** Function 01h

---

### INT 15h Function 01h

*CASSETTE*
*PC and PCjr only*

***TURN OFF TAPE DRIVE'S MOTOR***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 01h | CF set on error |

AH = 86h no cassette present
CF clear if successful

**Conflicts:** Amstrad (Chapter 7), MultiDOS Plus (Chapter 45), VMiX (Chapter 46)
**See also:** Cassette Function 00h

---

### INT 15h Function 02h
### *READ DATA*

*CASSETTE*
*PC and PCjr only*

**Registers at call:**
AH = 02h
CX = number of bytes to read
ES:BX -> buffer

**Return Registers:**
CF clear if successful
DX = number of bytes read
ES:BX -> byte following last byte read
CF set on error
AH = status (see Table 4-2)

**Conflicts:** Amstrad (Chapter 7), MultiDOS Plus (Chapter 45), VMiX (Chapter 46)
**See also:** Functions 00h and 03h

**4-2 Values for status:**

| | |
|---|---|
| 00h | successful |
| 01h | CRC error |
| 02h | bad tape signals |
| 04h | no data |
| 80h | invalid command |
| 86h | no cassette present |

---

### INT 15h Function 03h
### *WRITE DATA*

*CASSETTE*
*PC and PCjr only*

**Registers at call:**
AH = 03h
CX = number of bytes to write
ES:BX -> data buffer

**Return Registers:**
CF clear if successful
ES:BX -> byte following last byte written
CF set on error
AH = status (see Table 4-2)
CX = 0000h

**Conflicts:** Amstrad (Chapter 7), MultiDOS Plus (Chapter 45), VMiX (Chapter 46)
**See also:** Functions 00h and 02h

---

### INT 15h Function 04h
### *BUILD ABIOS SYSTEM PARAMETER TABLE*

*SYSTEM*
*PS/2*

**Registers at call:**
AH = 04h
ES:DI -> 32-byte results buffer for System Parameter Table (see Table 4-3)
DS = segment containing ABIOS RAM extensions (zero if none)

**Return Registers:**
AH = 00h success: results at ES:DI
CF set on failure

**Conflicts:** Amstrad (Chapter 7), MultiDOS Plus (Chapter 45), VMiX (Chapter 46)
**See also:** Functions 05h and C1h

**4-3 Format of ABIOS System Parameter Table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | FAR address of ABIOS Common Start Routine |
| 04h | DWORD | FAR address of ABIOS Interrupt Routine |
| 08h | DWORD | FAR address of ABIOS Time-out Routine |
| 0Ch | WORD | number of bytes of stack required by this ABIOS implementation |
| 0Eh | 16 BYTEs | reserved |
| 1Eh | WORD | number of entries in initialization table |

## INT 15h Function 05h
## *BUILD ABIOS INITIALIZATION TABLE*

*SYSTEM*
*PS/2*

**Registers at call:**
AH = 05h
ES:DI -> results buffer of length
(18h * Number_of_Entries) (see Table 4-4)
DS = segment containing ABIOS RAM extensions (zero if none)

**Return Registers:**
AH = 00h success: results at ES:DI
CF set on failure

**Conflicts:** Amstrad (Chapter 7), MultiDOS Plus (Chapter 45), VMiX (Chapter 46)
**See also:** Functions 04h and C1h

**4-4 Format of one entry of ABIOS Initialization Table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | device ID |
| 02h | WORD | number of Logical IDs |
| 04h | WORD | Device Block length (zero for ABIOS patch or extension) |
| 06h | DWORD | -> init routine for Device Block and Function Transfer Table |
| 0Ah | WORD | request block length |
| 0Ch | WORD | Function Transfer Table length (zero for a patch) |
| 0Eh | WORD | Data Pointers length (in Common Data Area) |
| 10h | BYTE | secondary device ID (hardware level this ABIOS ver supports) |
| 11h | BYTE | revision (device driver revision level this ABIOS supports) |
| 12h | 6 BYTEs | reserved |

## INT 15h Function 2010h
## *SETUP SYSREQ ROUTINE*

*OS HOOK*
*AT,XT286,PS/2-50+*

**Registers at call:**
AX = 2010h
*other registers, if any, unknown*

**Return Registers:**
*unknown*

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Function 2011h

## INT 15h Function 2011h
## *COMPLETION OF SYSREQ FUNCTION*

*OS HOOK*
*AT,XT286,PS/2-50+*

**Registers at call:**
AX = 2011h
*other registers, if any, unknown*

**Return Registers:**
*unknown*

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Function 2010h

## INT 15h Function 21h
## *POWER-ON SELF-TEST ERROR LOG*

*SYSTEM*
*PS/2 Model 50+*

**Registers at call:**
AH = 21h
AL = subfunction
  00h read POST log
  01h write POST log
    BH = device ID
    BL = error code

**Return Registers:**
CF set on error
AH = status (00h OK, 01h list full, 80h invalid cmd, 86h unsupported)
if function 00h:
  BX = number of error codes stored
  ES:DI -> error log

**Details:** The log is a series of words, the first byte of which identifies the error code and the second the device.

---

**INT 15h Function 22h** *SYSTEM*
***LOCATE ROM BASIC*** *later PS/2s*

**Registers at call:**
AH = 22h

**Return Registers:**
CF set on error
  AH = status (86h if function not supported)
CF clear on success
  AH = 00h
  ES:BX -> ROM BASIC

**Details:** If this function is not supported, ROM BASIC is at F600h:0000h.
**See also:** BASIC INT 86h and INT F0h (Chapter 54)

---

**INT 15h Function 2400h** *SYSTEM*
***DISABLE A20 GATE*** *later PS/2s*

**Registers at call:**
AX = 2400h

**Return Registers:**
CF clear if successful
  AH = 00h
CF set on error
  AH = status
    01h keyboard controller is in secure mode
    86h function not supported

**Details:** This function is also supported by Qualitas 386MAX v6.01+.
**See also:** Functions 2401h, 2402h, and 2403h

---

**INT 15h Function 2401h** *SYSTEM*
***ENABLE A20 GATE*** *later PS/2s*

**Registers at call:**
AX = 2401h

**Return Registers:**
CF clear if successful
  AH = 00h
CF set on error
  AH = status (see Function 2400h)
    01h keyboard controller is in secure mode
    86h function not supported

**Details:** This function is also supported by Qualitas 386MAX v6.01+.
**See also:** Functions 2400h and 2402h

---

**INT 15h Function 2402h** *SYSTEM*
***GET A20 GATE STATUS*** *later PS/2s*

**Registers at call:**
AX = 2402h

**Return Registers:**
CF clear if successful
  AH = 00h
  AL = current state (00h disabled, 01h enabled)
CF set on error
  AH = status (see Function 2400h)
    01h keyboard controller is in secure mode
    86h function not supported

**Details:** This function is also supported by Qualitas 386MAX v6.01+.
**See also:** Functions 2400h and 2401h

---

**INT 15h Function 2403h** *SYSTEM*
***QUERY A20 GATE SUPPORT*** *later PS/2s*

**Purpose:** Determine which methods of controlling the A20 gate are available.

**Registers at call:**
AX = 2403h

**Return Registers:**
CF clear if successful
    AH = 00h
    BX = status of A20 gate support
        bit 0: supported with bit 1 of I/O port 92h
        bit 1: supported on keyboard controller
        bit 15: additional data is available (location not yet defined)
CF set on error
    AH = status
        01h keyboard controller is in secure mode
        86h function not supported

**Details:** This function is also supported by Qualitas 386MAX v6.01+.
**See also:** Function 2402h

---

### INT 15h Function 40h
### *READ/MODIFY PROFILES*

*SYSTEM*
*IBM Convertible*

**Registers at call:**.
AH = 40h
AL = subfunction
    00h get system profile in CX and BX
    01h set system profile from CX and BX
    02h get internal modem profile in BX
    03h set internal modem profile from BX

**Return Registers:**
CF clear if successful
    AH = 00h
CF set on error
    AH = status (80h = profile execution failed)

**Conflicts:** Compaq extensions (Chapter 7)

---

### INT 15h Function 41h
### *WAIT ON EXTERNAL EVENT*

*SYSTEM*
*IBM Convertible and others*

**Registers at call:**
AH = 41h
AL = condition type
    bits 0-2: condition to wait for
        0 any external event
        1 compare and return if equal
        2 compare and return if not equal
        3 test and return if not zero
        4 test and return if zero
    bit 3: reserved
    bit 4: 1=port address, 0=user byte
    bits 5-7 reserved
BH = condition compare or mask value
BL = timeout value in 55 millisecond clock ticks (00h = no timeout)
DX = I/O port address if AL bit 4 set
ES:DI -> user byte if AL bit 4 clear

**Return Registers:**
none

**Details:** Call Function C0h and examine bit 3 of feature byte 1 to determine whether this function is supported.
**See also:** Functions 83h, 86h, and C0h

### INT 15h Function 42h
### *REQUEST POWER OFF*

*SYSTEM*
*IBM Convertible and HP 95LX*

**Registers at call:**
AH = 42h
AL = suspend type
    00h to use system profile
    01h to force suspend regardless of system profile

**Return Registers:**
none

**Details:** The HP 95LX apparently suspends regardless of the value in AL; on power-up, execution will resume following the instruction calling this function.
**Conflicts:** Compaq extensions (Chapter 7)
**See also:** Function 44h

### INT 15h Function 43h
### *READ SYSTEM STATUS*

*SYSTEM*
*IBM Convertible*

**Registers at call:**
AH = 43h

**Return Registers:**
AL = status bits (see Table 4-5)

**4-5 Bitfields for Convertible system status:**

| | |
|---|---|
| bit 0 | LCD detached |
| bit 1 | reserved |
| bit 2 | RS232/parallel adapter powered on |
| bit 3 | power activated by alarm |
| bit 4 | power low |
| bit 5 | standby power lost |
| bit 6 | external power in use |
| bit 7 | internal modem powered on |

### INT 15h Function 44h
### *(DE)ACTIVATE INTERNAL MODEM POWER*

*SYSTEM*
*IBM Convertible*

**Registers at call:**
AH = 44h
AL = new modem power state
    00h power off
    01h power on

**Return Registers:**
none

**See also:** Function 42h

### INT 15h Function 4Fh
### *KEYBOARD INTERCEPT*

*KEYBOARD*
*Callout*

**Registers at call:**
AH = 4Fh
AL = hardware scan code
CF set

**Return Registers:**
CF set
    AL = hardware scan code
CF clear
    scan code should be ignored

**Details:** Called by INT 09h handler to translate scan codes; the INT 09h code does not examine the scan code it reads from the keyboard until after this function returns. This permits software to rearrange the keyboard; for example, swapping the CapsLock and Control keys, or turning the right Shift key into Enter.

This function is supported on the AT model 3x9, XT2, XT286, Convertible, and PS/2 series.
**See also:** INT 09h, INT 15h Function C0h

### INT 15h Function 5101h
### *EXPANSION UNIT, RETURN CONFIGURATION NUMBER*

*SYSTEM*
*later PS/2s*

**Registers at call:**
AX = 5101h

**Return Registers:**
CF set if successful
    AH = 00h
    AL = current configuration number
        00h system unit only
        FFh configuration not recognized
    BX = status flag
        bits 0-14: reserved
        bit 15: additional data is available (location TBD
CF clear on error
    AH = status
        01h expansion unit is not present
        86h function not supported

**Details:** The CF convention is the reverse of the standard convention for this interrupt (perhaps a typo in the IBM BIOS Technical Reference?).
**Conflicts:** VMiX (Chapter 46)

### INT 15h Function 80h
### *DEVICE OPEN (AT,XT286,PS)*

*OS HOOK*
*Callout*

**Purpose:** Notify interested programs that a device is now in use.

**Registers at call:.**
AH = 80h
BX = device ID
CX = process ID
CF clear

**Return Registers:**
CF clear if successful
    AH = 00h
CF set on error
    AH = status
        80h invalid command (PC,PCjr)
        86h function not supported (XT)

**Details:** This function should be hooked by a multitasker which wishes to keep track of device ownership; the default BIOS handler merely returns successfully.
**See also:** Functions 81h and 82h

### INT 15h Function 81h
### *DEVICE CLOSE*

*OS HOOK*
*Callout*

**Purpose:** Notify interested programs that a device is no longer being used.

**Registers at call:**
AH = 81h
BX = device ID
CX = process ID
CF clear

**Return Registers:**
CF clear if successful
    AH = 00h
CF set on error
    AH = status (see Function 80h)

**Details:** This function should be hooked by a multitasker which wishes to keep track of device ownership; the default BIOS handler merely returns successfully.
**See also:** Functions 80h and 82h

### INT 15h Function 82h
### *PROGRAM TERMINATION*

*OS HOOK*
*Callout*

**Purpose:** Notify interested programs that the devices opened by a particular program are no longer in use.

**Registers at call:**
AH = 82h

BX = process ID
CF clear

**Return Registers:**
CF clear if successful
    AH = 00h
CF set on error
    AH = status (see Function 80h)

**Details:** Closes all devices opened by the given process ID with function 80h.

This function should be hooked by a multitasker which wishes to keep track of device ownership; the default BIOS handler merely returns successfully.

**See also:** Functions 80h and 81h

---

### INT 15h Function 83h
### *SET EVENT WAIT INTERVAL*

*BIOS*
*AT, PS/2 Model 50+*

**Registers at call:**
AH = 83h
AL = subfunction
    00h set interval
        CX:DX = microseconds to delay ES:BX -> byte whose high bit is to be set at end of interval
    01h cancel wait interval

**Return Registers:**
CF set on error or function already busy
    AH = status
        80h invalid command (PC,PCjr)
        86h function not supported (XT and later)
CF clear if successful

**Details:** The resolution of the wait period is 977 microseconds on most systems because most BIOSes use the 1/1024 second fast interrupt from the AT real-time clock chip which is available on INT 70h.

The IBM AT 1/10/84 BIOS ignores AL and always performs subfunction 00h.

**See also:** Functions 41h and 86h, IRQ8 INT 70h (Chapter 3)

---

### INT 15h Function 84h
### *JOYSTICK SUPPORT*

*BIOS*
*XT after 11/8/82, AT, XT/286, PS/2*

**Registers at call:**
AH = 84h
DX = subfunction

**Return Registers:**
CF set on error
    AH = status (see Function 83h)
CF clear if successful

| | |
|---|---|
| 0000h read joystick switches | AL bits 7-4 = switch settings |
| 0001h read positions of joysticks | AX = X position of joystick A<br>BX = Y position of joystick A<br>CX = X position of joystick B<br>DX = Y position of joystick B |

**Details:** If no game port is installed, subfunction 0000h returns AL=00h (all switches open) and subfunction 0001h returns AX=BX=CX=DX=0000h; some BIOSes return AH=86h instead.

A 250kOhm joystick typically returns values in the range 0000h-01A0h.

---

### INT 15h Function 85h
### *SysRq KEY ACTIVITY*

*OS HOOK*
*AT, PS/2 Callout*

**Purpose:** Notify any interested programs (particularly the operating system) of activity on the SysRq key.

**Registers at call:**.
AH = 85h
AL = action
    00h SysRq key pressed

01h SysRq key released
CF clear

**Return Registers:**
CF clear if successful
AH = 00h
CF set on error
AH = status (see Function 83h)

**Details:** This function is called by the keyboard decode routine.

The default handler simply returns successfully; programs which wish to monitor the SysRq key must hook this call.

**See also:** IRQ1 INT 09h (Chapter 3)

---

### INT 15h Function 86h
### *WAIT*

*BIOS*
*AT, PS/2*

**Registers at call:**
AH = 86h
CX:DX = interval in microseconds

**Return Registers:**
CF clear if successful (wait interval elapsed)
CF set on error or Function 83h wait already in progress
AH = status (see Function 83h)

**Details:** The resolution of the wait period is 977 microseconds on most systems because most BIOSes use the 1/1024 second fast interrupt from the AT real-time clock chip which is available on INT 70h.

**See also:** Function 83h, SND INT 1Ah Function FF01h (Chapter 22), IRQ8 INT 70h (Chapter 3)

---

### INT 15h Function 87h
### *COPY EXTENDED MEMORY*

*SYSTEM*
*80286 and higher processors*

**Registers at call:**
AH = 87h
CX = number of words to copy (max 8000h)
ES:SI -> global descriptor table (Table 4-6)

**Return Registers:**
CF set on error
CF clear if successful
AH = status
00h source copied into destination
01h parity error
02h interrupt error
03h address line 20 gating failed
80h invalid command (PC,PCjr)
86h unsupported function (XT,PS30)

**Details:** The copy is performed in protected mode with interrupts disabled by the default BIOS handler; many 386 memory managers perform the copy with interrupts enabled.

This function is incompatible with the OS/2 compatibility box.

**See also:** Functions 88h and 89h

**4-6 Format of global descriptor table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 16 BYTEs | zeros |
| 10h | WORD | source segment length in bytes (2*CX-1 or greater) |
| 12h | 3 BYTEs | 24-bit linear source address, low byte first |
| 15h | BYTE | source segment access rights (93h) |
| 16h | WORD | zero |
| 18h | WORD | destination segment length in bytes (2*CX-1 or greater) |
| 1Ah | 3 BYTEs | 24-bit linear destination address, low byte first |
| 1Dh | BYTE | destination segment access rights (93h) |
| 1Eh | 18 BYTEs | zeros |

---

### INT 15h Function 88h
### *GET EXTENDED MEMORY SIZE*

*SYSTEM*
*80286 and higher processors*

**Registers at call:**
AH = 88h

**Return Registers:**
CF clear if successful
  AX = number of contiguous KB starting at absolute address 100000h
CF set on error
  AH = status
    80h invalid command (PC,PCjr)
    86h unsupported function (XT,PS30)

**Details:** TSRs which wish to allocate extended memory to themselves often hook this call, and return a reduced memory size. They are then free to use the memory between the new and old sizes at will.

The standard BIOS only returns memory between 1MB and 16MB; use Function C7h for memory beyond 16MB.

**See also:** Functions 87h and C7h

---

## INT 15h Function 89h
## *SWITCH TO PROTECTED MODE*

*SYSTEM*
*80286 and higher processors*

**Registers at call:**
AH = 89h
BL = interrupt number of IRQ0
  (IRQ1-7 use next 7 interrupts)
BH = interrupt number of IRQ8
  (IRQ9-F use next 7 interrupts)
ES:SI -> GDT for protected mode (see Table 4-7)
CX = offset into protected-mode CS to jump to

**Return Registers:**
CF set on error
  AH = FFh error enabling address line 20
CF clear if successful
  AH = 00h
  in protected mode at specified address

**Details:** BL and BH must be multiples of 8.

**See also:** Functions 87h and 88h, VCPI INT 67h Function DE0Ch (Chapter 29)

**4-7 Format of GDT:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | QWORD | null descriptor (initialize to zeros) |
| 08h | QWORD | GDT descriptor |
| 10h | QWORD | IDT descriptor |
| 18h | QWORD | DS |
| 20h | QWORD | ES |
| 28h | QWORD | SS |
| 30h | QWORD | CS |
| 38h | QWORD | uninitialized, used to build descriptor for BIOS CS |

---

## INT 15h Function 90h
## *DEVICE BUSY*

*OS HOOK*
*AT, PS/2 Callout*

**Purpose:** Notify interested programs that a device is busy with an operation and some other activity can be performed until the operation completes.

**Registers at call:**
AH = 90h
AL = device type (see Table 4-8)
ES:BX -> request block for type codes 80h through BFh
CF clear

**Return Registers:**
CF set if wait time satisfied
CF clear if driver must perform wait
  AH = 00h

**Details:** Type codes are allocated as follows: 00h-7Fh non-reentrant devices, OS must arbitrate access; 80h-BFh reentrant devices, ES:BX points to a unique control block; C0h-FFh wait-only calls, no complementary INT 15h Function 91h call.

The floppy and hard disk BIOS code uses this call to implement a timeout; for device types 00h and 01h, a return of CF set means that the timeout expired before the disk responded.

This function should be hooked by a multitasker to allow other tasks to execute while the BIOS is waiting for I/O completion; the default handler merely returns with AH=00h and CF clear.

**See also:** Function 91h, Disk INT 13h Function 00h (Chapter 10), Printer INT 17h Function 00h (Chapter 15), Tandy sound INT 1Ah Function 83h (Chapter 22)

**4-8 Values for device type:**

| | |
|---|---|
| 00h | disk |
| 01h | diskette |
| 02h | keyboard |
| 03h | PS/2 pointing device |
| 21h | waiting for keyboard input (Phoenix BIOS) |
| 80h | network |
| FBh | digital sound (Tandy) |
| FCh | disk reset (PS/2) |
| FDh | diskette motor start |
| FEh | printer |

---

### INT 15h Function 91h
### *DEVICE POST*

*OS HOOK*
*AT, PS/2 Callout*

**Purpose:** Notify interested programs that a device has completed its current operation and is ready for another.

**Registers at call:**
AH = 91h
AL = device type (see Table 4-8)
ES:BX -> request block for type codes
  80h through BFh
CF clear

**Return Registers:**
AH = 00h

**Details:** This function should be hooked by a multitasker to allow other tasks to execute while the BIOS is waiting for I/O completion; the default handler merely returns with AH=00h and CF clear.

**See also:** Function 90h

---

### INT 15h Function C0h
### *GET CONFIGURATION*

*SYSTEM*
*XT after 1/10/86, AT model 3x9, and newer*

**Registers at call:**
AH = C0h

**Return Registers:**
CF set if BIOS doesn't support call
CF clear on success
    ES:BX -> ROM table (see Table 4-9)
AH = status
    00h successful
    86h unsupported function

**Details:** The 1/10/86 XT BIOS returns an incorrect value for the feature byte.

The configuration table is at F000h:E6F5h in 100% compatible BIOSes.

Dell machines contain the signature "DELL " or "Dell " at absolute FE076h and a model byte (see Table 4-11) at absolute address FE845h.

Tandy 1000 machines contain 21h in the byte at F000h:C000h.

Some AST machines contain the string "COPYRIGHT AST RESEARCH " one byte past the end of the configuration table.

The Phoenix 386 BIOS contains a second version and date string (presumably the last modification for that OEM version) beginning at F000h:FFD8h, with each byte doubled (so that both ROM chips contain the complete information).
**See also:** Functions C7h, C9h, and D1h

**4-9 Format of ROM configuration table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of bytes following |
| 02h | BYTE | model (see Table 4-10) |
| 03h | BYTE | submodel (see Table 4-10) |
| 04h | BYTE | BIOS revision: 0 for first release, 1 for 2nd, etc. |
| 05h | BYTE | feature byte 1: |
| | | bit 7: DMA channel 3 used by hard disk BIOS |
| | | bit 6: 2nd 8259 installed |
| | | bit 5: Real-Time Clock installed |
| | | bit 4: INT 15h Function 4Fh called upon INT 09h |
| | | bit 3: wait for external event (INT 15h Function 41h) supported |
| | | bit 2: extended BIOS area allocated (usually at top of RAM) |
| | | bit 1: bus is Micro Channel instead of ISA |
| | | bit 0: system has dual bus (Micro Channel + ISA) |
| 06h | BYTE | feature byte 2: |
| | | bit 7: reserved |
| | | bit 6: INT 16h Function 09h (keyboard functionality) supported |
| | | bit 5: INT 15h Function C6h (get POS data) supported |
| | | bit 4: INT 15h Function C7h (return memory map info) supported |
| | | bit 3: INT 15h Function C8h (en/disable CPU functions) supported |
| | | bit 2: non-8042 keyboard controller |
| | | bit 1: data streaming supported |
| | | bit 0: reserved |
| 07h | BYTE | feature byte 3: |
| | | bits 7-5 reserved |
| | | bit 4: *unknown* (set on 1992 PS/1's, N51SX, CL57SX, 35SX?, 40SX?) |
| | | bit 3: SCSI subsystem supported on system board |
| | | bit 2: information panel installed |
| | | bit 1: IML (Initial Machine Load) system |
| | | bit 0: SCSI supported in IML |
| 08h | BYTE | feature byte 4: |
| | | bit 7: *unknown* (set on N51SX, CL57SX) |
| | | bits 6-4: reserved |
| | | bit 3: *unknown* (set on some 1992 PS/1's, 35SX, 40SX) |
| | | bits 2-1: reserved |
| | | bit 0: *unknown* (set on N51SX, CL57SX, 57SX) |
| 09h | BYTE | feature byte 5: |
| | | reserved (0) (IBM) |
| | | *unknown* (08h) (Phoenix 386 v1.10) |
| *AWARD BIOS* | | |
| 0Ah | N BYTEs | AWARD copyright notice |

*Phoenix BIOS*

| | | |
|---|---|---|
| 0Ah | BYTE | *unknown* (00h) |
| 0Bh | BYTE | major version |
| 0Ch | BYTE | minor version (BCD) |
| 0Dh | 4 BYTEs | ASCIZ string "PTL " (Phoenix Technologies Ltd) |

*Quadram Quad386*

| | | |
|---|---|---|
| 0Ah | 17 BYTEs | ASCII signature string "Quadram Quad386XT " |

### 4-10 Values for model/submodel/revision:

| *Model* | *Submdl* | *Revis* | *BIOS date* | | *System* |
|---|---|---|---|---|---|
| FFh | * | * | 04/24/81 | | PC (original) |
| FFh | * | * | 10/19/81 | | PC (some bugfixes) |
| FFh | * | * | 10/27/82 | | PC (HD, 640K, EGA support) |
| FFh | 46h | *** | | | Olivetti M15 |
| FEh | * | * | 08/16/82 | | PC XT |
| FEh | * | * | 11/08/82 | | PC XT and Portable |
| FEh | 43h | *** | | | Olivetti M240 |
| FEh | A6h | ? | | | Quadram Quad386 |
| FDh | * | * | 06/01/83 | | PCjr |
| FCh | * | * | 01/10/84 | | AT models 068,099 6 MHz 20MB |
| FCh | 00h | 01h | 06/10/85 | | AT model 239 6 MHz 30MB |
| FCh | 00h | <> 01h | | | 7531/2 Industrial AT |
| FCh | 01h | 00h | 11/15/85 | | AT models 319,339 8 MHz, Enh Keyb, 3.5 |
| FCh | 01h | 00h | 09/17/87 | | Tandy 3000 |
| FCh | 01h | 00h | 01/15&88 | | Toshiba T5200/100 |
| FCh | 01h | 00h | 12/26*89 | | Toshiba T1200/XE |
| FCh | 01h | 30h | | | Tandy 3000NL |
| FCh | 01h | ? | | | Compaq 286/386 |
| FCh | 02h | 00h | 04/21/86 | | PC XT-286 |
| FCh | 04h | 00h | 02/13/87 | ** | PS/2 Model 50 (10 MHz/1 ws 286) |
| FCh | 04h | 03h | 04/18/88 | | PS/2 Model 50Z (10 MHz/0 ws 286) |
| FCh | 05h | 00h | 02/13/87 | ** | PS/2 Model 60 (10 MHz 286) |
| FCh | 06h | ? | | | 7552 "Gearbox " |
| FCh | 08h | *** | | | Epson, unknown model |
| FCh | 09h | 00h | | | PS/2 Model 25 (10 MHz 286) |
| FCh | 09h | 02h | 06/28/89 | | PS/2 Model 30-286 |
| FCh | 0Bh | 00h | 02/16/90 | | PS/1 Model 2011 (10 MHz 286) |
| FCh | 30h | *** | | | Epson, unknown model |
| FCh | 31h | *** | | | Epson, unknown model |
| FCh | 33h | *** | | | Epson, unknown model |
| FCh | 42h | *** | | | Olivetti M280 |
| FCh | 45h | *** | | | Olivetti M380 (XP 1, XP3, XP 5) |
| FCh | 48h | *** | | | Olivetti M290 |
| FCh | 4Fh | *** | | | Olivetti M250 |
| FCh | 50h | *** | | | Olivetti M380 (XP 7) |
| FCh | 51h | *** | | | Olivetti PCS286 |
| FCh | 52h | *** | | | Olivetti M300 |
| FCh | 81h | 00h | 01/15/88 | | Phoenix 386 BIOS v1.10 10a |
| FBh | 00h | 01h | 01/10/86 | | PC XT, Enh Keyb, 3.5 " support |

| | | | | | |
|---|---|---|---|---|---|
| FBh | 00h | 02h | 05/09/86 | | PC XT |
| FBh | 4Ch | *** | | | Olivetti M200 |
| FAh | 00h | 00h | 09/02/86 | | PS/2 Model 30 (8 MHz 8086) |
| FAh | 00h | 01h | 12/12/86 | | PS/2 Model 30 |
| FAh | 01h | 00h | | | PS/2 Model 25/25L (8 MHz 8086) |
| FAh | 4Eh | *** | | | Olivetti M111 |
| F9h | 00h | 00h | 09/13/85 | | PC Convertible |
| F8h | 00h | 00h | 03/30/87 | ** | PS/2 Model 80 (16MHz 386) |
| F8h | 01h | 00h | 10/07/87 | | PS/2 Model 80 (20MHz 386) |
| F8h | 04h | 02h | 04/11/88 | | PS/2 Model 70 20MHz, type 2 system brd |
| F8h | 04h | 03h | 03/17/89 | | PS/2 Model 70 20MHz, type 2 system brd |
| F8h | 09h | 00h | | | PS/2 Model 70 16MHz, type 1 system brd |
| F8h | 09h | 02h | 04/11/88 | | PS/2 Model 70 some models |
| F8h | 09h | 03h | 03/17/89 | | PS/2 Model 70 some models |
| F8h | 0Bh | 00h | 01/18/89 | | PS/2 Model P70 (8573-121) typ 2 sys brd |
| F8h | 0Bh | 02h | 12/16/89 | | PS/2 Model P70 ?? |
| F8h | 0Ch | 00h | 11/02/88 | | PS/2 Model 55SX (16 MHz 386SX) |
| F8h | 0Dh | 00h | | | PS/2 Model 70 25MHz, type 3 system brd |
| F8h | 11h | 00h | 10/01/90 | | PS/2 Model 90 (25 MHz 486) |
| F8h | 13h | 00h | 10/01/90 | | PS/2 Model 90 (33 MHz 486) |
| F8h | 14h | 00h | 10/01/90 | | PS/2 Model 90-AK9 (25 MHz 486) |
| F8h | 16h | 00h | 10/01/90 | | PS/2 Model 90-AKD (33 MHz 486) |
| F8h | 19h | 05h | | | PS/2 Model 35/35LS or 40 (20 MHz 386SX) |
| F8h | 1Bh | 00h | 10/02/89 | | PS/2 Model 70-486 (25 MHz 486) |
| F8h | 1Ch | 00h | 02/08/90 | | PS/2 Model 65-121 (16 MHz 386SX) |
| F8h | 1Eh | 00h | 02/08/90 | | PS/2 Model 55LS (16 MHz 386SX) |
| F8h | 23h | 01h | | | PS/2 Model L40 (20 MHz 386SX) |
| F8h | 25h | 06h | | | PS/2 Model M57 (20 MHz 386SLC) |
| F8h | 26h | 01h | | | PS/2 Model 57 (20 MHz 386SX) |
| F8h | 2Ah | 00h | | | PS/2 Model 95 (50 MHz 486) |
| F8h | 2Bh | 00h | | | PS/2 Model 90 (50 MHz 486) |
| F8h | 2Ch | 01h | | | PS/2 Model 95 (20 MHz 486SX) |
| F8h | 2Dh | 00h | | | PS/2 Model 90 (20 MHz 486SX) |
| F8h | 2Eh | 01h | | | PS/2 Model 95 (20 MHz 486SX + 487SX) |
| F8h | 2Fh | 00h | | | PS/2 Model 90 (20 MHz 486SX + 487SX) |
| F8h | 30h | 00h | | | PS/1 Model 2121 (16 MHz 386SX) |
| F8h | 50h | 00h | | | PS/2 Model P70 (8573) (16 MHz 386) |
| F8h | 50h | 01h | 12/16/89 | | PS/2 Model P70 (8570-031) |
| F8h | 52h | 00h | | | PS/2 Model P75 (33 MHz 486) |
| F8h | 61h | *** | | | Olivetti P500 |
| F8h | 62h | *** | | | Olivetti P800 |
| F8h | 80h | 00h | | | PS/2 Model 80 (25 MHz 386) |
| F8h | 80h | 01h | 11/21/89 | | PS/2 Model 80-A21 |
| F8h | ? | ? | | | PS/2 Model 90 (25 MHz 486SX) |
| F8h | ? | ? | | | PS/2 Model 95 (25 MHz 486SX) |
| F8h | ? | ? | | | PS/2 Model 90 (25 MHz 486SX + 487SX) |
| F8h | ? | ? | | | PS/2 Model 95 (25 MHz 486SX + 487SX) |
| E1h | ? | ? | | | *unknown* (checked for by DOS4GW.EXE) |
| 9Ah | * | * | | | Compaq XT/Compaq Plus |

| | | | |
|---|---|---|---|
| 30h | ? | ? | Sperry PC |
| 2Dh | * | * | Compaq PC/Compaq Deskpro |
| ? | 56h | ? | Olivetti, unknown model |
| ? | 74h | ? | Olivetti, unknown model |

**NOTES**

- * This BIOS call is not implemented in these early versions. Read Model byte at F000h:FFFEh and BIOS date at F000h:FFF5h.
- ** These BIOS versions require the DASDDRVR.SYS patches.
- *** These Olivetti and Epson machines store the submodel in the byte at F000h:FFFDh.

**4-11 Values for Dell model byte:**

| | |
|---|---|
| 02h | Dell 200 |
| 03h | Dell 300 |
| 05h | Dell 220 |
| 06h | Dell 310 |
| 07h | Dell 325 |
| 09h | Dell 310A |
| 0Ah | Dell 316 |
| 0Bh | Dell 220E |
| 0Ch | Dell 210 |
| 0Dh | Dell 316SX |
| 0Eh | Dell 316LT |
| 0Fh | Dell 320LX |
| 11h | Dell 425E |

---

### INT 15h Function C1h

### *RETURN EXTENDED-BIOS DATA-AREA SEGMENT ADDRESS*

*SYSTEM*
*PS/2*

**Registers at call:**
AH = C1h

**Return Registers:**
CF set on error
CF clear if successful
    ES = segment of data area

**See also:** Function 04h

---

### INT 15h Functions C200h-C207h

### *POINTING DEVICE BIOS INTERFACE—see Chapter 21*

*SYSTEM*

---

### INT 15h Function C3h

### *ENABLE/DISABLE WATCHDOG TIMEOUT*

*SYSTEM*
*PS/2 Model 50+*

**Registers at call:**
AH = C3h
AL = function
    00h disable PS/2 watchdog timer
    01h enable PS/2 watchdog timer
        BX = timer counter (0001h-00FFh)
    02h disable Gearbox system
    03h enable Gearbox system

**Return Registers:**
CF set on error
CF clear if successful

**Details:** The watchdog timer generates an NMI (INT 02h, Chapter 3).

---

### INT 15h Function C4h

### *PROGRAMMABLE OPTION SELECT*

*SYSTEM*
*PS/2 Model 50+*

**Registers at call:**.
AH = C4h
AL = subfunction

00h return base POS register address
01h enable selected slot for setup
BL = slot number (1 to 8)
02h disable setup for all slots

**Return Registers:**
CF set on error
DX = base POS register address
(if subfunction 00h)

**See also:** Function C6h

---

### INT 15h Function C5h
### *ROM BIOS TRACING CALLOUT*

*PS/2-30/286 and newer*
*(undocumented) Callout*

**Registers at call:**
AH = C5h
AL = interrupt being invoked
01h INT 19h
02h INT 14h
03h INT 16h
04h INT 40h (floppy INT 13h)
05h INT 17h
06h INT 10h
07h INT 12h
08h INT 11h
09h INT 1Ah

**Return Registers:**
all registers except AX must be preserved

**Details:** This function is called as the very first action of the indicated ROM BIOS interrupt handlers on the PS/2 Models 30/286, 50Z, and 95.

The default handler does nothing and returns CF clear for the above subfunctions, CF set and AH=86h for all other subfunctions.

The value of AX passed to the original interrupt handler is pushed on stack immediately prior to call.

---

### INT 15h Function C6h
### *GET POS DATA*

*later PS/2 models*
*(undocumented)*

**Registers at call:**
AH = C6h
*other registers, if any, unknown*

**Return Registers:**
*unknown*

**Details:** This function is referenced by name and number in some IBM BIOS manuals.

IBM reports that "there are a number of problems with this call " and does not recommend its use.

**See also:** Function C4h

---

### INT 15h Function C7h
### *RETURN MEMORY-MAP INFORMATION*

*SYSTEM*
*later PS/2s*

**Registers at call:**
AH = C7h
DS:SI -> user supplied memory map table (see Table 4-12)

**Return Registers:**
CF set on error, clear if successful

**Details:** Call Function C0h and examine bit 4 of feature byte 2 to check if this function is supported.

**See also:** Functions C0h, C9h, and D1h

**4-12 Format of memory-map table structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | length of table (excluding this word) |
| 02h | DWORD | local memory between 1M and 16M, in 1K blocks |
| 06h | DWORD | local memory between 16M and 4G, in 1K blocks |
| 0Ah | DWORD | system memory between 1M and 16M, in 1K blocks |

| | | |
|---|---|---|
| 0Eh | DWORD | system memory between 16M and 4G, in 1K blocks |
| 12h | DWORD | cacheable memory between 1M and 16M, in 1K blocks |
| 16h | DWORD | cacheable memory between 16M and 4G, in 1K blocks |
| 1Ah | DWORD | 1K blocks before start of non-system memory between 1M and 16M |
| 1Eh | DWORD | 1K blocks before start of non-system memory between 16M and 4G |
| 22h | 2 DWORDs | reserved |

---

### INT 15h Function C8h
### *ENABLE/DISABLE PROCESSOR FUNCTIONS*

*SYSTEM*
*PS/2 Model 70 and newer*

**Registers at call:**
AH = C8h
AL = function
    00h disable cache or (90 and 95) disable cache L1
    01h enable cache or (90 and 95) enable cache L1
    **—models 90 and 95 only—**
    02h disable cache L2
    03h enable cache L2
    04h disable both caches
    05h enable both caches
    06h return status of both caches
    07h-FFh Reserved

**Return Registers:**
CF set on error
CF clear if successful
    AH = status (see Table 4-13)
For subfunction 06h only:
    BH = status of cache L2
        00h enabled
        01h disabled or not installed
    BL = status of cache L1 (same codes as BH)

**Details:** This function is supported by at least PS/2 Models 70, 70/486, 80-A21, 90, and 95.

Call Function C0h and examine bit 3 of feature byte 2 to check if this function is supported.

On a 486 system, any external caches must be disabled when the on-chip cache (L1) is disabled.

**See also:** Function C0h

**4-13 Values for status:**

| | |
|---|---|
| 00h | operation successful |
| 01h | function choice (in AL) is invalid |
| 02h | NVRAM data is invalid |
| 03h | cache test error |
| 04h | (90 and 95 only) cannot perform operation requested due to state of other cache (see note above) |
| 05h | no L2 cache is present |

---

### INT 15h Function C9h
### *GET CPU TYPE AND MASK REVISION*

*newer PS/2; various BIOSes*

**Registers at call:**
AH = C9h
AL = 10h (may be required on some non-PS BIOSes)

**Return Registers:**
CF clear if successful
    AH = 00h
    CH = CPU type
        03h 80386DX or clone
        04h 80486
        23h 80386SX or clone
    CL = mask revision (stepping level)
        80386:
            03h B1
            05h D0
            08h D1/D2/E1

80386SX:
    08h D1
CF set on error
    AH = status (80h,86h = function not supported)

**Details:** The BIOS must save DX at startup in order to be able to support this call; PS/2 Models 56, 57, 90, and 95 are known to support it.

The PS/2 BIOS merely reads CMOS locations 190h (type) and 191h (revision).

---

### INT 15h Functions CAh-CDh
### *RESERVED*

*PS/2 Model 95 (undocumented)*

**Registers at call:**.
AH = CAh through CDh
*other registers, if any, unknown*

**Return Registers:**
*unknown*

---

### INT 15h Function CEh
### *ALLOCATE DMA ARBITRATION LEVEL*

*later PS/2s*

**Registers at call:**
AH = CEh
BL = operation
    00h-0Eh arbitration level to be allocated
    0Fh-FFh reserved
AL = option byte
    bit 7-1: reserved (0)
    bit 0: no DMA channel required for requested arbitration level

**Return Registers:**
CF set on error
    AH = status (80h,86h = function not supported)
CF clear on success
    AL = channel number
        00h-07h channel number allocated for the arbitration level
        08h-FEh reserved
        FFh no channel requested for arbitration level
    AH = status
        00h success
        01h arbitration level not available
        02h channel not available
        03h invalid arbitration level passed

**Details:** Arbitration level 00h has the highest priority, 0Eh the lowest.

To perform a DMA transfer operation, be sure to call this function first, and call Function CFh afterward. Failure to use this function can cause unpredictable results.

**See also:** Function CFh

---

### INT 15h Function CFh
### *DEALLOCATE DMA ARBITRATION LEVEL*

*later PS/2s*

**Registers at call:**
AH = CFh
BL = arbitration level to be deallocated (see Function CEh)

**Return Registers:**
CF set on error
    AH = status (80h,86h = function not supported)
CF clear on success
    AH = status
        00h success
        04h arbitration level not allocated

**See also:** Function CEh

---

### INT 15h Function D0h
### *RESERVED*

*later PS/2s*

**Registers at call:**
AH = D0h
*other registers, if any, unknown*

**Return Registers:**
*unknown*

## INT 15h Function D100h

*later PS/2s*

### *GET NUMBER OF DEVICE DESCRIPTOR TABLE (DDT) ENTRIES*

**Registers at call:**
AX = D100h
DX = 0000h (reserved, must set to 0)

**Return Registers:**
BL = size of one DDT entry, in bytes
CX = number of DDT entries
AH = return code (see Table 4-14)
CF set on error
CF clear on success

**See also:** Functions C0h, C7h, C9h, D101h, D102h, D103h, and D104h

**4-14 Values for return code:**

| | |
|---|---|
| 00h | success |
| 01h | requested DDT entry not found |
| 02h | DDT data not valid |
| 86h | function not supported |

## INT 15h Function D101h

*later PS/2s*

### *RETURN DEVICE DESCRIPTOR TABLE (DDT) ENTRY BY NUMBER*

**Registers at call:**
AX = D101h
BX = number of requested entry
(starting with 1)
DX = 0000h (reserved, must be set to 0)
ES:DI -> buffer to contain DDT entry
(see Table 4-15)

**Return Registers:**
AH = return code (see Table 4-14)
CF set on error
CF clear on success
ES:DI buffer filled with DDT entry

**See also:** Functions C0h, C7h, C9h, D100h, D102h, and D104h

**4-15 Format of DDT:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | bits 7-4: reserved (set to 0)<br>bits 3-0: slot of device (0 = system board) |
| 01h | BYTE | bits 7-4: second interrupt for this device (0 = none)<br>bits 3-0: first interrupt for this device (0 = none) |
| 02h | BYTE | bits 7-4: second arbitration level for this device<br>bits 3-0: first arbitration level for this device |
| 03h | WORD | DDT indicators (see Table 4-16) |
| 05h | BYTE | reserved (0) |
| 06h | WORD | device ID (0 = none) |
| 08h | WORD | starting address of first I/O block (0 = none) |
| 0Ah | WORD | starting address of second I/O block (0 = none) |
| OCh | WORD | starting address of third I/O block (0 = none) |
| OEh | DWORD | start of first non-system memory block (0 = none) |
| 12h | WORD | size of first non-system memory block (in kilobytes) |
| 14h | DWORD | start of second non-system memory block (0 = none) |
| 18h | WORD | size of second non-system memory block (in kilobytes) |
| 1Ah | BYTE | implementation identifier of the device |
| 1Bh | BYTE | implementation revision level of the device |

**Note:** I/O block addresses and non-system memory addresses are listed in ascending order in each DDT entry.

**4-16 Bitfields for DDT indicators:**

| | |
|---|---|
| bit 15 | reserved (0) |
| bit 14 | second arbitration level exists |
| bit 13 | first arbitration level exists |
| bit 12 | serial interface is RS-422 |
| bit 11 | not address limited |
| bit 10 | DMA channel used |
| bit 9 | second arbitration level can be shared |
| bit 8 | first arbitration level can be shared |
| bits 7-0 | reserved (0) |

---

### INT 15h Function D102h

*later PS/2s*

***RETURN DEVICE DESCRIPTOR TABLE (DDT) ENTRY BY I/O ADDRSS***

**Purpose:** The DDT is searched from the specified entry for the I/O port in CX, and the first entry in which it is found is returned.

**Registers at call:**

AX = D102h

BX = entry number at which to start searching

CX = requested I/O port address

DX = 0000h (reserved, must be set to 0)

ES:DI -> buffer to contain DDT entry (see Table 4-15)

**Return Registers:**

AH = return code (see Table 4-14)

BX = DDT entry number where I/O port was found, or total entries plus 1 if port was not found.

CF set on error

CF clear on success

ES:DI buffer filled with DDT entry

**See also:** Functions C0h, C7h, C9h, D100h, D101h, D103h, and D104h

---

### INT 15h Function D103h

*later PS/2s*

***RETURN ENTIRE DDT***

**Registers at call:**

AX = D103h

DX = 0000h (reserved, must be set to 0)

ES:DI -> buffer to contain DDT entry (see Table 4-15)

**Return Registers:**

AH = return code (see Table 4-14)

CF set on error

CF clear on success

ES:DI buffer filled with DDT entry

**See also:** Functions C0h, C7h, C9h, D100h, and D104h

---

### INT 15h Function D104h

*later PS/2s*

***RETURN DEVICE DESCRIPTOR TABLE (DDT) ENTRY BY DEVICE ID***

**Purpose:** The DDT is searched from the specified entry for the device ID in CX, and the first entry in which it is found is returned.

**Registers at call:**

AX = D104h

BX = entry number at which to start searching

CX = requested device ID

DX = 0000h (reserved, must be set to 0)

ES:DI -> buffer to contain DDT entry (see Table 4-15)

**Return Registers:**

AH = return code (see Table 4-14)

BX = DDT entry number where device ID was found, or total entries plus 1 if port was not found.

CF set on error

CF clear on success

ES:DI buffer filled with DDT entry

**See also:** Functions C0h, C7h, C9h, D100h, D101h, and D103h

### INT 15h Functions D2h and D3h

*later PS/2s*

***RESERVED***

**Registers at call:**
AH = D2h and D3h
*other registers, if any, unknown*

**Return Registers:**
*unknown*

### INT 15h Function D4h

*later PS/2s*

***GET PHYSICAL FIXED DISK DRIVE NUMBER (SELECTABLE BOOT)***

**Registers at call:**
AH = D4h
DL = logical fixed disk drive number

**Return Registers:**
AH = return code
  00h success
  01h specified logical drive number is invalid
  80h function not supported (on PCjr and PC)
  86h function not supported
CF set on error
CF clear on success
  AL = physical fixed disk drive number

### INT 15h Function D5h

*later PS/2s*

***RESERVED***

**Registers at call:**
AH = D5h
*other registers, if any, unknown*

**Return Registers:**
*unknown*

### INT 15h Function D600h, Subfunction 00h

*later PS/2s*

***READ BOOT DEVICE ID***

**Registers at call:**
AX = D600h
BL = 00h
DX = device ID

**Return Registers:**
CF clear if successful
  AH = 00h
CF set on error
  AH = status (86h if function not supported)

**See also:** Function D600h Subfunction 01h, Function D601h Subfunction 00h, and Function D602h

### INT 15h Function D600h, Subfunction 01h

*later PS/2s*

***WRITE BOOT DEVICE ID***

**Registers at call:**
AX = D600h
BL = 01h
DX = device ID

**Return Registers:**
CF clear on success
  AH = 00h
CF set on error
  AH = status (86h if function not supported)

**See also:** Function D600h Subfunction 00h and Function D601h Subfunction 01h

### INT 15h Function D601h, Subfunction 00h

*later PS/2s*

***READ BOOT DEVICE KEY***

**Registers at call:**
AX = D601h
BL = 00h
DX = device ID

**Return Registers:**
CF clear on success
  AH = 00h
CF set on error
  AH = status (86h if function not supported)

**See also:** Function D600h Subfunction 00h, Function D601h Subfunction 01h, and Function D602h

### INT 15h Function D601h, Subfunction 01h
*later PS/2s*

***WRITE BOOT DEVICE KEY***

**Registers at call:**
AX = D601h
BL = 01h
DX = device ID

**Return Registers:**
CF clear on success
    AH = 00h
CF set on error
    AH = status (86h if function not supported)

**See also:** Function D600h Subfunction 01h and Function D601h Subfunction 00h

---

### INT 15h Function D602h
*later PS/2s*

***QUERY BOOT REFERENCE PARTITION***

**Registers at call:**
AX = D602h

**Return Registers:**
CF clear on success
    AH = 00h
    AL = status of reference-partition boot request
        00h boot not requested
        01h boot requested
CF set on error
    AH = status (86h for function not supported)

**See also:** Function D601h Subfunction 00h

---

### INT 16h Function 00h
*KEYBOARD*

***GET KEYSTROKE***

**Registers at call:**
AH = 00h

**Return Registers:**
AH = BIOS scan code
AL = ASCII character

**Details:** On extended keyboards, this function discards any extended keystrokes, returning only when a non-extended keystroke is available.

The BIOS scan code is usually, but not always, the same as the hardware scan code processed by INT 09h. It is the same for ASCII keystrokes and most unshifted special keys (F-keys, arrow keys, etc.), but differs for shifted special keys.

**See also:** Functions 01h, 05h, 10h, and 20h; NEC PC-9800 INT 18h Function 00h (Chapter 6)

---

### INT 16h Function 01h
*KEYBOARD*

***CHECK FOR KEYSTROKE***

**Registers at call:**
AH = 01h

**Return Registers:**
ZF set if no keystroke available
ZF clear if keystroke available
    AH = BIOS scan code
    AL = ASCII character

**Details:** If a keystroke is present, it is not removed from the keyboard buffer; however, any extended keystrokes which are not compatible with 83/84- key keyboards are removed in the process of checking whether a non-extended keystroke is available.

**See also:** Functions 00h, 11h, and 21h; NEC PC-9800 INT 18h Function 01h (Chapter 6)

---

### INT 16h Function 02h
*KEYBOARD*

***GET SHIFT FLAGS***

**Registers at call:**
AH = 02h

**Return Registers:**
AL = shift flags (see Table 4-17)

**See also:** Functions 12h and 22h, Emulaser INT 17h Function 0Dh (Chapter 15), NEC PC-9800 INT 18h Function 02h (Chapter 6)

**4-17 Bitfields for shift flags:**

| | |
|---|---|
| bit 7 | Insert active |
| bit 6 | CapsLock active |
| bit 5 | NumLock active |
| bit 4 | ScrollLock active |
| bit 3 | Alt key pressed (either Alt on 101/102-key keyboards) |
| bit 2 | Ctrl key pressed (either Ctrl on 101/102-key keyboards) |
| bit 1 | left shift key pressed |
| bit 0 | right shift key pressed |

---

### INT 16h Function 03h
### *SET TYPEMATIC RATE AND DELAY*

*KEYBOARD*

**Registers at call:**
AH = 03h
AL = subfunction
  00h set default delay and rate (PCjr and some PS/2)
  01h increase delay before repeat (PCjr)
  02h decrease repeat rate by factor of 2 (PCjr)
  03h increase delay and decrease repeat rate (PCjr)
  04h turn off typematic repeat (PCjr and some PS/2)
  05h set repeat rate and delay (AT,PS)
    BH = delay value (00h = 250ms to 03h = 1000ms)
    BL = repeat rate (00h=30/sec to 0Ch=10/sec [default] to 1Fh=2/sec)
  06h get current typematic rate and delay (newer PS/2s)

**Return Registers:**
BL = repeat rate (see above)
BH = delay (see above)

**Details:** Use INT 16h Function 09h to determine whether some of the subfunctions are supported.

**See also:** INT 16h Function 09h

---

### INT 16h Function 04h
### *SET KEYCLICK*

*KEYBOARD*
*PCjr only*

**Registers at call:**
AH = 04h
AL = keyclick state
  00h off, 01h on

**Return Registers:**
none

**See also:** Function 03h

---

### INT 16h Function 05h
### *STORE KEYSTROKE IN KEYBOARD BUFFER*

*KEYBOARD*
*AT/PS with enhanced kbd only*

**Registers at call:**
AH = 05h
CH = scan code
CL = ASCII character

**Return Registers:**
AL = status
  00h successful
  01h keyboard buffer full

**Details:** Under DESQview, the following "keystrokes " invoke the following actions when they are read from the keyboard buffer:

- 38FBh or FB00h switch to next window (only if main menu popped up)
- 38FCh or FC00h pop up DESQview main menu
- 38FEh or FE00h close the current window
- 38FFh or FF00h pop up DESQview learn menu.

**Conflicts:** PCjr Function 05h below

**See also:** Function 00h, FAKEY Function 71h (Chapter 13), KBUF Function FFh (Chapter 13), DESQview INT 15h Function DE10h (Chapter 44)

---

### INT 16h Function 05h
### *SELECT KEYBOARD LAYOUT*

*KEYBOARD*
*PCjr only*

**Registers at call:**
AH = 05h
AL = function
    01h set keyboard layout to French
    02h set keyboard layout to German
    03h set keyboard layout to Italian
    04h set keyboard layout to Spanish
    05h set keyboard layout to UK
    80h check if function supported

**Return Registers:**
AL Ö 80h if supported

**Return Registers:**
*unknown*

**Details:** Called by DOS 3.2 KEYBxx.COM.
**Conflicts:** AT/PS2 Function 05h above
**See also:** Functions 92h and A2h (Chapter 27)

---

### INT 16h Function 09h
### *GET KEYBOARD FUNCTIONALITY*

*KEYBOARD*
*newer BIOSes*

**Registers at call:**
AH = 09h

**Return Registers:**
AL = supported keyboard functions (see Table 4-18)

**Details:** This function is only available if bit 6 of the second feature byte returned by INT 15h Function C0h is set (see Table 4-9).
**See also:** Functions 03h, 0Ah, 10h, 11h, 12h, 20h, 21h, and22h; INT 15h Function C0h

**4-18 Bitfields for supported keyboard functions:**

| | |
|---|---|
| bit 7 | reserved |
| bit 6 | INT 16h Functions 20h-22h supported (122-key keyboard support) |
| bit 5 | INT 16h Functions 10h-12h supported (enhanced keyboard support) |
| bit 4 | INT 16h Function 0Ah supported |
| bit 3 | INT 16h Function 0306h supported |
| bit 2 | INT 16h Function 0305h supported |
| bit 1 | INT 16h Function 0304h supported |
| bit 0 | INT 16h Function 0300h supported |

---

### INT 16h Function 0Ah
### *GET KEYBOARD ID*

*KEYBOARD*
*newer BIOSes*

**Registers at call:**
AH = 0Ah

**Return Registers:**
BX = keyboard ID or 0000h if no keyboard attached

**Details:** Check return value from Function 09h to determine whether this function is supported.
**See also:** Function 09h

---

### INT 16h Function 10h
### *GET ENHANCED KEYSTROKE*

*KEYBOARD*
*enhanced keyboard support only*

**Registers at call:**
AH = 10h

**Return Registers:**
AH = BIOS scan code
AL = ASCII character

**Details:** If no keystroke is available, this function waits until one is placed in the keyboard buffer.

The BIOS scan code is usually, but not always, the same as the hardware scan code processed by INT 09h. It is the same for ASCII keystrokes and most unshifted special keys (F-keys, arrow keys, etc.), but differs for shifted special keys.

Unlike Function 00h, this function does not discard extended keystrokes.

INT 16h Function 09h can be used to determine whether this function is supported, but only on later model PS/2s.

**See also:** Functions 00h, 09h, 11h, and 20h

---

### INT 16h Function 11h
### *CHECK FOR ENHANCED KEYSTROKE*

*KEYBOARD*
*enhanced keyboard support only*

**Registers at call:**
AH = 11h

**Return Registers:**
ZF set if no keystroke available
ZF clear if keystroke available
  AH = BIOS scan code
  AL = ASCII character

**Details:** If a keystroke is available, it is not removed from the keyboard buffer.

Unlike Function 01h, this function does not discard extended keystrokes.

Some versions of the IBM BIOS Technical Reference erroneously report that CF is returned instead of ZF.

INT 16h Function 09h can be used to determine whether this function is supported, but only on later model PS/2s.

**See also:** Functions 01h, 09h, 10h, and 21h

---

### INT 16h Function 12h
### *GET EXTENDED SHIFT STATES*

*KEYBOARD*
*enhanced keyboard support only*

**Registers at call:**
AH = 12h

**Return Registers:**
AL = shift flags 1 (same as returned by Function 02h) (see Table 4-19)
AH = shift flags 2 (see Table 4-20)

**Details:** AL bit 3 set only for left Alt key on many machines.

AH bits 7 through 4 are always clear on a Compaq SLT/286.

INT 16h Function 09h can be used to determine whether this function is supported, but only on later model PS/2s.

**See also:** Functions 02h, 09h, and 22h; AX PC Function 51h (Chapter 6), Emulaser INT 17h Function 0Dh (Chapter 15)

**4-19 Bitfields for shift flags 1:**

| | |
|---|---|
| bit 7 | Insert active |
| bit 6 | CapsLock active |
| bit 5 | NumLock active |
| bit 4 | ScrollLock active |
| bit 3 | Alt key pressed (either Alt on 101/102-key keyboards) |
| bit 2 | Ctrl key pressed (either Ctrl on 101/102-key keyboards) |
| bit 1 | left shift key pressed |
| bit 0 | right shift key pressed |

**4-20 Bitfields for shift flags 2:**

| | |
|---|---|
| bit 7 | SysRq key pressed |
| bit 6 | CapsLock pressed |
| bit 5 | NumLock pressed |
| bit 4 | ScrollLock pressed |
| bit 3 | right Alt key pressed |
| bit 2 | right Ctrl key pressed |
| bit 1 | left Alt key pressed |
| bit 0 | left Ctrl key pressed |

---

## INT 16h Function 20h

*KEYBOARD*

### *GET 122-KEY KEYSTROKE*

*122-key keyboard support only*

**Registers at call:**
AH = 20h

**Return Registers:**
AH = BIOS scan code (see Function 10h for details)
AL = ASCII character

**Details:** Use Function 09h to determine whether this function is supported.

**See also:** Functions 00h, 09h, 10h, 21h, and 22h

---

## INT 16h Function 21h

*KEYBOARD*

### *CHECK FOR 122-KEY KEYSTROKE*

*122-key keyboard support only*

**Registers at call:**
AH = 21h

**Return Registers:**
ZF set if no keystroke available
ZF clear if keystroke available
    AH = BIOS scan code
    AL = ASCII character

**Details:** Use Function 09h to determine whether this function is supported.

Some versions of the IBM BIOS Technical Reference erroneously report that CF is returned instead of ZF.

**See also:** Functions 01h, 09h, 11h, 20h, and 21h

---

## INT 16h Function 22h

*KEYBOARD*

### *GET 122-KEY SHIFT STATUS*

*122-key keyboard support only*

**Registers at call:**
AH = 22h

**Return Registers:**
AL = shift flags 1 (see Table 4-19)
AH = shift flags 2 (see Table 4-20)

**Details:** Use Function 09h to determine whether this function is supported.

**See also:** Functions 02h, 09h, 12h, 20h, and 21h

---

## INT 17h Functions 00h-02h

*BIOS*

### *PRINTER SERVICES—see Chapter 15*

---

## INT 18h

*BIOS*

### *DISKLESS BOOT HOOK (START CASSETTE BASIC)*

*Callout*

**Purpose:** Called when there is no bootable disk available to the system.

**Details:** Only PCs produced by IBM contain BASIC in ROM, so the action is unpredictable on compatibles; this interrupt often reboots the system, and often has no effect at all.

Network cards with their own BIOS can hook this interrupt to allow a diskless boot off the network (even when a hard disk is present if none of the partitions is marked as the boot partition).

**Conflicts:** NEC PC-9800 (Chapter 6)

**See also:** NetBIOS INT 86h (NI)

---

## INT 19h

*SYSTEM*

### *BOOTSTRAP LOADER*

**Purpose:** This interrupt reboots the system without clearing memory or restoring interrupt vectors. Because interrupt vectors are preserved, this interrupt usually causes a system hang if any TSRs have hooked vectors from 00h through 1Ch, particularly INT 08h.

**Details:** Usually, the BIOS will try to read sector 1, head 0, track 0 from drive A: to 0000h:7C00h. If this fails, and a hard disk is installed, the BIOS will read sector 1, head 0, track 0 of the first hard disk. This sector should contain a master bootstrap loader and a partition table. After loading the master boot sector at 0000h:7C00h, the master bootstrap loader is given control. It will scan the partition table for an active partition, and will then load the operating system's bootstrap loader (contained in the first sector of the active partition) and give it control.

True IBM PCs and most clones issue an INT 18h if neither floppy nor hard disk have a valid boot sector.

To accomplish a warm boot equivalent to Ctrl-Alt-Del, store 1234h in 0040h:0072h and jump to FFFFh:0000h. For a cold boot equivalent to a reset, store 0000h at 0040h:0072h before jumping. However, simply jumping to the reset address may cause data loss with delayed-write disk caches, so it is advisable to issue an INT 21h Function 0Dh (see Chapter 24) and an INT 15h Function 4Fh (see above) simulating Ctrl-Alt-Del to force a cache flush.

VDISK.SYS hooks this interrupt to allow applications to find out how much extended memory has been used by VDISKs (see Table 4-21). DOS 3.3+ PRINT hooks INT 19h but does not set up a correct VDISK header block at the beginning of its INT 19h handler segment, thus causing some programs to overwrite extended memory which is already in use.

The default handler is at F000h:E6F2h for 100% compatible BIOSes.

MS-DOS 3.2+ hangs on booting (even from floppy) if the hard disk contains extended partitions which point at each other in a loop, since it will never find the end of the linked list of extended partitions.

Some adapter cards do not properly reinitialize themselves either on a software reset or a hardware reset signal, and thus require that the power be cycled off and then back on.

**See also:** INT 18h, FOSSIL INT 14h Function 17h (Chapter 12)

**4-21 Format of VDISK header block (at beginning of INT 19h handler's segment):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 18 BYTEs | n/a (for VDISK.SYS, the device driver header) |
| 12h | 11 BYTEs | signature string "VDISK Vn.m " for VDISK.SYS version n.m |
| 1Dh | 15 BYTEs | n/a |
| 2Ch | 3 BYTEs | linear address of first byte of available extended memory |

**4-22 Format of hard disk master boot sector:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 446 BYTEs | Master bootstrap loader code |
| 1BEh | 16 BYTEs | partition record for partition 1 (see Table 4-23) |
| 1CEh | 16 BYTEs | partition record for partition 2 |
| 1DEh | 16 BYTEs | partition record for partition 3 |
| 1EEh | 16 BYTEs | partition record for partition 4 |
| 1FEh | WORD | signature, AA55h indicates valid boot block |

**4-23 Format of partition record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | boot indicator (80h = active partition) |
| 01h | BYTE | partition start head |
| 02h | BYTE | partition start sector (bits 0-5) |
| 03h | BYTE | partition start track (bits 8,9 in bits 6,7 of sector) |
| 04h | BYTE | operating system indicator (see Table 4-24) |
| 05h | BYTE | partition end head |
| 06h | BYTE | partition end sector (bits 0-5) |
| 07h | BYTE | partition end track (bits 8,9 in bits 6,7 of sector) |
| 08h | DWORD | sectors preceding partition |
| 0Ch | DWORD | length of partition in sectors |

**4-24 Values for operating system indicator:**

| | |
|---|---|
| 00h | empty |
| 01h | DOS 12-bit FAT |
| 02h | XENIX root file system |
| 03h | XENIX /usr file system (obsolete) |

| | |
|---|---|
| 04h | DOS 16-bit FAT |
| 05h | DOS 3.3+ extended partition |
| 06h | DOS 3.31+ Large File System |
| 07h | QNX |
| 07h | OS/2 HPFS |
| 07h | Advanced Unix |
| 08h | AIX bootable partition, SplitDrive |
| 09h | AIX data partition |
| 09h | Coherent filesystem |
| 0Ah | OS/2 Boot Manager |
| 0Ah | OPUS |
| 0Ah | Coherent swap partition |
| 10h | OPUS |
| 24h | NEC MS-DOS 3.x |
| 40h | VENIX 80286 |
| 50h | Disk Manager, read-only partition |
| 51h | Disk Manager, read/write partition |
| 51h | *Novell* |
| 52h | CP/M |
| 52h | Microport System V/386 |
| 56h | GoldenBow VFeature |
| 61h | SpeedStor |
| 63h | Unix SysV/386, 386/ix |
| 63h | Mach, MtXinu BSD 4.3 on Mach |
| 63h | GNU HURD |
| 64h | Novell NetWare |
| 65h | Novell NetWare (3.11) |
| 75h | PC/IX |
| 80h | Minix v1.1 - 1.4a |
| 81h | Minix v1.4b+ |
| 81h | Linux |
| 81h | Mitac Advanced Disk Manager |
| 82h | Linux Swap partition (planned) |
| 84h | OS/2-renumbered type 04h partition (related to hiding DOS C: drive) |
| 93h | Amoeba file system |
| 94h | Amoeba bad block table |
| B7h | BSDI file system (secondarily swap) |
| B8h | BSDI swap partition (secondarily file system) |
| C1h | DR-DOS 6.0 LOGIN.EXE-secured 12-bit FAT partition |
| C4h | DR-DOS 6.0 LOGIN.EXE-secured 16-bit FAT partition |
| C6h | DR-DOS 6.0 LOGIN.EXE-secured Huge partition |
| DBh | CP/M, Concurrent CP/M, Concurrent DOS |
| DBh | CTOS (Convergent Technologies OS) |
| E1h | SpeedStor 12-bit FAT extended partition |
| E4h | SpeedStor 16-bit FAT extended partition |
| F2h | DOS 3.3+ secondary |
| FEh | LANstep |
| FFh | Xenix bad block table |

---

**INT 1Ah Function 00h** *TIME*

***GET SYSTEM TIME***

**Registers at call:**
AH = 00h

**Return Registers:**
CX:DX = number of clock ticks since midnight
AL = midnight flag, nonzero if midnight passed since time last read

**Details:** There are approximately 18.2 clock ticks per second, 1800B0h per 24 hours.

IBM and many clone BIOSes set the flag for AL rather than incrementing it, leading to loss of a day if two consecutive midnights pass without a request for the time (e.g. if the system is on but idle).

**See also:** Functions 01h and 02h; DOS INT 21h Function 2Ch (Chapter 24), INT 62h Function 0099h (Chapter 52)

---

**INT 1Ah Function 01h** *TIME*

***SET SYSTEM TIME***

**Registers at call:**
AH = 01h
CX:DX = number of clock ticks since midnight

**See also:** Functions 00h and 03h; DOS INT 21h Function 2Dh (Chapter 24)

---

**INT 1Ah Function 02h** *TIME*
*AT,XT/286,PS/2*

***GET REAL-TIME CLOCK TIME***

**Registers at call:**
AH = 02h

**Return Registers:**
CF clear if successful
  CH = hour (BCD)
  CL = minutes (BCD)
  DH = seconds (BCD)
  DL = daylight savings flag (00h standard time, 01h daylight time)
CF set on error (i.e. clock not running or in middle of update)

**See also:** Function 00h

---

**INT 1Ah Function 03h** *TIME*
*AT,XT/286,PS/2*

***SET REAL-TIME CLOCK TIME***

**Registers at call:**
AH = 03h
CH = hour (BCD)
CL = minutes (BCD)
DH = seconds (BCD)
DL = daylight savings flag (00h standard time, 01h daylight time)

**Return Registers:**
none

**See also:** Function 01h

---

**INT 1Ah Function 04h** *TIME*
*AT,XT/286,PS/2*

***GET REAL-TIME CLOCK DATE***

**Registers at call:**
AH = 04h

**Return Registers:**
CF clear if successful
  CH = century (BCD)
  CL = year (BCD)
  DH = month (BCD)
  DL = day (BCD)
CF set on error

**See also:** Functions 02h and 05h, DOS INT 21h Function 2Ah (Chapter 24)

**INT 1Ah Function 00h**

### INT 1Ah Function 05h

*TIME*

***SET REAL-TIME CLOCK DATE***

*AT,XT/286,PS/2*

**Registers at call:**
AH = 05h
CH = century (BCD)
CL = year (BCD)
DH = month (BCD)
DL = day (BCD)

**Return Registers:**
none

**See also:** Function 04h,DOS INT 21h Function 2Bh (Chapter 24)

### INT 1Ah Function 06h

*TIME*

***SET ALARM***

*AT,XT/286,PS/2*

**Registers at call:**
AH = 06h
CH = hour (BCD)
CL = minutes (BCD)
DH = seconds (BCD)

**Return Registers:**
CF set on error (alarm already set or clock stopped for update)
CF clear if successful

**Details:** The alarm occurs every 24 hours until turned off, invoking INT 4Ah each time; many BIOSes disable the alarm at the RTC after the first occurrence.

**See also:** Function 07h, INT 4Ah

### INT 1Ah Function 07h

*TIME*

***CANCEL ALARM***

*AT,XT/286,PS/2*

**Registers at call:**
AH = 07h

**Return Registers:**
alarm disabled

**Details:** Does not disable the real-time clock's IRQ.

**See also:** Function 06h, INT 70h

### INT 1Ah Function 08h

*TIME*

***SET RTC ACTIVATED POWER ON MODE***

*IBM Convertible*

**Registers at call:**
AH = 08h
CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD

**Return Registers:**
none

**See also:** Function 09h

### INT 1Ah Function 09h

*TIME*

***READ RTC ALARM TIME AND STATUS***

*IBM Convertible, PS/2 Model 30*

**Registers at call:**
AH = 09h

**Return Registers:**
CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD
DL = alarm status
    00h alarm not enabled
    01h alarm enabled but will not power up system
    02h alarm will power up system

**See also:** Function 08h

### INT 1Ah Function 0Ah
### *READ SYSTEM-TIMER DAY COUNTER*

*TIME*
*XT2,PS/2*

**Registers at call:**
AH = 0Ah

**Return Registers:**
CF set on error
CF clear if successful
CX = count of days since Jan 1,1980

**See also:** Functions 04h and 0Bh

### INT 1Ah Function 0Bh
### *SET SYSTEM-TIMER DAY COUNTER*

*TIME*
*XT2,PS/2*

**Registers at call:**
AH = 0Bh
CX = count of days since Jan 1,1980

**Return Registers:**
CF set on error
CF clear if successful

**See also:** Functions 05h and 0Ah

### INT 1Bh
### *CONTROL-BREAK HANDLER*

*KEYBOARD*
*Callout*

**Purpose:** This interrupt is automatically called when INT 09h determines that Control-Break has been pressed.

**Details:** INT 1Bh normally points to a short routine in DOS which sets the Ctrl-C flag, thus invoking INT 23h the next time DOS checks for Ctrl-C.

**See also:** DOS INT 23h (Chapter 24)

### INT 1Ch
### *SYSTEM TIMER TICK*

*SYSTEM*
*Callout*

**Purpose:** This interrupt is automatically called on each clock tick by the INT 08h handler.

**Details:** This is the preferred interrupt to chain when a program needs to be invoked regularly.
This interrupt is not available on NEC 9800-series PCs.

**See also:** IRQ0 INT 08h (Chapter 3)

### INT 1Dh
### *VIDEO PARAMETER TABLES—see Chapter 8*

*SYSTEM*

### INT 1Eh
### *DISKETTE PARAMETERS—see Chapter 10*

*SYSTEM*

### INT 1Fh
### *GRAPHICS FONT—see Chapter 8*

*SYSTEM*

### INT 4Ah
### *USER ALARM HANDLER*

*SYSTEM*
*Callback*

**Purpose:** This interrupt is invoked by the BIOS when a real-time clock alarm occurs; an application may use it to perform an action at a predetermined time.

**Details:** This interrupt is called from within a hardware interrupt handler, so all usual precautions against reentering DOS must be taken.

**See also:** INT 1Ah Function 06h

### INT 6Ch
### *SYSTEM RESUME VECTOR*

*IBM Convertible*

**INT FEh** *AT/XT286/PS50+*

### *DESTROYED BY RETURN FROM PROTECTED MODE*

**Details:** The ROM BIOS uses 0030h:0100h as the initial stack on startup, which is the last fourth of the interrupt vector table. If the processor is returned to real mode via a hardware reset (the only possibility on an 80286, though there are a number of ways of generating one), then the BIOS startup code stacks three words on its scratch stack before determining that a return to real mode has been requested. As a result, INT FEh and INT FFh are corrupted.

**See also:** INT FFh

**INT FFh** *AT/XT286/PS50+*

### *DESTROYED BY RETURN FROM PROTECTED MODE*

**Details:** (see INT FEh).

**See also:** INT FEh

## CHAPTER 5

# Expansion Bus BIOSes

The original IBM PC expansion bus (and its later augmentation to the 16-bit PC/AT expansion bus) is simplistic enough that software need not manipulate it or even be aware of its existence. More recent peripheral buses, however, have added a plethora of configurable features, requiring some form of software control. This chapter describes the standard interfaces provided for three expansion buses: EISA, Intel's PCI local bus, and the PCMCIA "memory card" bus.

In addition to the buses covered in this chapter, the IBM PS/2 system BIOS for Microchannel machines contains some calls related to that bus; those are described in Chapter 4.

## EISA System ROM

EISA, the Enhanced Industry Standard Architecture expansion bus, is a superset of IBM's original IBM PC/AT expansion bus which was created in response to IBM's proprietary Microchannel bus. It features many of the enhancements made by the Microchannel while remaining backward-compatible with PC/AT peripheral cards.

The standard installation check for an EISA System ROM is to test for the signature "EISA" at absolute address F000h:FFD9h. There are no known interrupt conflicts for the EISA System ROM.

---

### INT 15h Function D800h
*EISA SYSTEM ROM*

***READ SLOT CONFIGURATION INFORMATION***

**Registers at call:**
AX = D800h
CL = slot number (including embedded and virtual)

**Return Registers:**
CF clear if successful
    AH = 00h
CF set on error
    AH = error code (80h,82h,83h,86h,87h) (see Table 5-1)
AL = bit flags
    bit 7: set if duplicate IDs
    bit 6: set if product ID readable
    bits 4,5: slot type (00=expansion, 01=embedded, 10=virtual device)
    bits 0-3: duplicate ID number if bit 7 set
BH = major revision level of configuration utility
BL = minor revision level of configuration utility
CX = checksum of configuration file
DH = number of device functions
DL = combined function information byte
SI:DI = 4-byte compressed ID (DI = bytes 0 and 1, SI = bytes 2 and 3)

**Details:** Call with AL=80h if using 32-bit CS addressing mode instead of 16-bit.

**See also:** Functions D801h, D804h, and D880h

**5-1 EISA System ROM error codes**

| | |
|---|---|
| 80h | invalid slot number |
| 81h | invalid function number |
| 82h | EISA CMOS corrupt |
| 83h | empty slot |
| 84h | error clearing CMOS |
| 85h | EISA CMOS is full |
| 86h | invalid BIOS-FW function call |
| 87h | invalid system configuration |
| 88h | config utility version not supported |

---

### INT 15h Function D801h
*EISA SYSTEM ROM*

***READ FUNCTION CONFIGURATION INFORMATION***

**Registers at call:**
AX = D801h
CH = function number to read
CL = slot number (including embedded and virtual)
DS:SI -> 320-byte buffer for standard configuration data block

**Return Registers:**
CF clear if successful
    AH = 00h
    DS:SI buffer filled
CF set on error
    AH = error code (80h-83h,86h,87h) (see Table 5-1)
BX destroyed

**Details:** Call with AL=81h if using 32-bit CS addressing mode instead of 16-bit.

**See also:** Functions D800h and D881h

---

### INT 15h Function D802h

*EISA SYSTEM ROM*

***CLEAR NONVOLATILE MEMORY (EISA CMOS)***

**Registers at call:**
AX = D802h
BH = EISA config utility major revision level
BL = EISA config utility minor revision level

**Return Registers:**
CF clear if successful
AH = 00h
CF set on error
AH = error code (84h,86h,88h) (see Table 5-1)

**Details:** Call with AL=82h if using 32-bit CS addressing mode instead of 16-bit. **See also:** Functions D803h and D882h

---

### INT 15h Function D803h

*EISA SYSTEM ROM*

***WRITE NONVOLATILE MEMORY***

**Registers at call:**
AX = D803h
CX = length of data structure (0000h= empty slot), includes two bytes for config file checksum
DS:SI -> configuration data

**Return Registers:**
CF clear if successful
AH = 00h
CF set on error
AH = error code (84h-86h)

**Details:** Call with AL=83h if using 32-bit CS addressing mode instead of 16-bit.. **See also:** Functions D802h and D883h

---

### INT 15h Function D804h

*EISA SYSTEM ROM*

***READ PHYSICAL SLOT***

**Registers at call:**
AX = D804h
CL = slot number (including embedded and virtual)

**Return Registers:**
CF clear if successful
AH = 00h
CF set on error
AH = error code (80h,83h,86h) (see Table 5-1)
SI:DI = 4-byte compressed ID (DI = bytes 0 and 1, SI = bytes 2 and 3)

**Details:** Call with AL=84h if using 32-bit CS addressing mode instead of 16-bit. **See also:** Functions D800h and D884h

---

### INT 15h Functions D880h to D884h

*EISA SYSTEM ROM*

***32-bit CS ADDRESSING MODE CALLS***

**Registers at call:**
AX = D880h to D884h
other registers as appropriate for AL=00h to 04h

**Return Registers:**
as appropriate for AL=00h to 04h

**Details:** These functions are identical to Functions D800h through D804h, except that they should be called when using 32-bit CS addressing mode (pointers use ESI rather than SI as offset) instead of 16-bit addressing mode.
**See also:** Functions D800h, D801h, D802h, D803h, and D804h

## Intel PCI BIOS

The PCI bus is Intel's answer to the VESA Local Bus (VL-Bus). While it overcomes a number of the limitations of the VL-Bus, such as supporting more than three slots and supporting higher speeds, it has taken much longer to reach the market and is in danger of missing its window of opportunity now that the VL-Bus has become commonplace. There are no known interrupt conflicts with the Intel PCI BIOS.

---

### INT 1Ah Function B101h

*Intel PCI BIOS v2.0c*

***INSTALLATION CHECK***

**Registers at call:**
AX = B101h

**Return Registers:**
AH = 00h if installed
CF clear
EDX = 20494350h (' ICP')
AL = PCI hardware characteristics (see Table 5-2)
BH = PCI interface level major version (BCD)
BL = PCI interface level minor version (BCD)
CL = number of last PCI bus in system
EDI = physical address of protected-mode entry point (see Table 5-3)
EAX, EBX, ECX, and EDX may be modified
all other flags (except IF) may be modified

**Details:** This function may require up to 1024 bytes of stack; it will not enable interrupts if they were disabled before making the call. **See also:** Functions B102h and B181h

#### 5-2 PCI Hardware Characteristics

| | |
|---|---|
| bit 0 | configuration space access mechanism 1 supported |
| bit 1 | configuration space access mechanism 2 supported |
| bits 2-3 | reserved |
| bit 4 | Special Cycle generation mechanism 1 supported |
| bit 5 | Special Cycle generation mechanism 2 supported |
| bits 6-7 | reserved |

#### 5-3 Protected-mode Entry Point

**Registers at call:**
as for real/V86-mode INT call
CS = ring 0 descriptor with access to full address space

**Return Registers:**
Same as for real/V86-mode call

---

### INT 1Ah Function B102h

*Intel PCI BIOS v2.0c*

### *FIND PCI DEVICE*

**Registers at call:**
AX = B102h
CX = device ID
DX = vendor ID
SI = device index (0-n)

**Return Registers:**
CF clear if successful
CF set on error
AH = status (00h,83h,86h) (see Table 5-4)
if successful:
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 func)
EAX, EBX, ECX, and EDX may be modified
All other flags (except IF) may be modified

**Details:** This function may require up to 1024 bytes of stack; it will not enable interrupts if they were disabled before making the call. Device ID FFFFh may be reserved as a wildcard in future implementations. The meanings of BL and BH on return may be exchanged in future implementations. All devices sharing a single vendor ID and device ID may be enumerated by incrementing SI from 0 until error 86h is returned. **See also:** Functions B103h and B182h

**5-4 Intel PCI BIOS Error Codes:**

| | | | |
|---|---|---|---|
| 00h | successful | 86h | device not found |
| 81h | unsupported function | 87h | bad register number |
| 83h | bad vendor ID | | |

---

### INT 1Ah Function B103h

*Intel PCI BIOS v2.0c*

### *FIND PCI CLASS CODE*

**Registers at call:**
AX = B103h
ECX = class code (bits 23-0)
SI = device index (0-n)

**Return Registers:**
CF clear if successful
CF set on error
AH = status (00h,86h) (see Table 5-4)
if successful:
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 func)
EAX, EBX, ECX, and EDX may be modified
All other flags (except IF) may be modified

**Details:** This function may require up to 1024 bytes of stack; it will not enable interrupts if they were disabled before making the call. The meanings of BL and BH on return may be exchanged in future implementations. All devices sharing the same Class Code may be enumerated by incrementing SI from 0 until error 86h is returned. **See also:** Function B183h

---

### INT 1Ah Function B106h

*Intel PCI BIOS v2.0c*

### *PCI BUS-SPECIFIC OPERATIONS*

**Purpose:** Perform an implementation-specific command.

**Registers at call:**
AX = B106h
BL = bus number
EDX = Special Cycle data

**Return Registers:**
CF clear if successful
CF set on error
AH = status (00h,81h) (see Table 5-4)
EAX, EBX, ECX, and EDX may be modified
All other flags (except IF) may be modified

**Details:** This function may require up to 1024 bytes of stack; it will not enable interrupts if they were disabled before making the call. **See also:** Function B186h

---

### INT 1Ah Function B108h

*Intel PCI BIOS v2.0c*

### *READ CONFIGURATION BYTE*

**Purpose:** Retrieve a single byte from the PCI bus configuration registers.

**Registers at call:**
AX = B108h
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 function)
DI = register number (0000h-00FFh)

**Return Registers:**
CF clear if successful
CL = byte read
CF set on error
AH = status (00h,87h) (see Table 5-4)
EAX, EBX, ECX, and EDX may be modified
All other flags (except IF) may be modified

**Details:** This function may require up to 1024 bytes of stack; it will not enable interrupts if they were disabled before making the call. The meanings of BL and BH on entry may be exchanged in future implementations. **See also:** Function B188h

---

### INT 1Ah Function B109h

*Intel PCI BIOS v2.0c*

### *READ CONFIGURATION WORD*

**Purpose:** Retrieve a word from the PCI bus configuration registers.

**Registers at call:**
AX = B109h
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 function)
DI = register number (0000h-00FFh)

**Return Registers:**
CF clear if successful
CX = word read
CF set on error
AH = status (00h,87h) (see Table 5-4)
EAX, EBX, ECX, and EDX may be modified
All other flags (except IF) may be modified

### INT 1Ah Function B10Ah
*Intel PCI BIOS v2.0c*

### *READ CONFIGURATION DWORD*

**Purpose:** Retrieve a doubleword from the PCI bus configuration registers.

**Registers at call:**
AX = B10Ah
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 function)
DI = register number (0000h-00FFh)

**Return Registers:**
CF clear if successful
ECX = dword read
CF set on error
AH = status (00h,87h) (see Table 5-4)
EAX, EBX, ECX, and EDX may be modified
All other flags (except IF) may be modified

**Details:** This function may require up to 1024 bytes of stack; it will not enable interrupts if they were disabled before making the call. The meanings of BL and BH on entry may be exchanged in future implementations. **See also:** Function B18Ah

### INT 1Ah Function B10Bh
*Intel PCI BIOS v2.0c*

### *WRITE CONFIGURATION BYTE*

**Registers at call:**
AX = B10Bh
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 function)
DI = register number (0000h-00FFh)
CL = byte to write

**Return Registers:**
CF clear if successful
CF set on error
AH = status (00h,87h) (see Table 5-4)
EAX, EBX, ECX, and EDX may be modified
All other flags (except IF) may be modified

**Details:** This function may require up to 1024 bytes of stack; it will not enable interrupts if they were disabled before making the call. The meanings of BL and BH on entry may be exchanged in future implementations. See also: Function B18Bh

### INT 1Ah Function B10Ch
*Intel PCI BIOS v2.0c*

### *WRITE CONFIGURATION WORD*

**Registers at call:**
AX = B10Ch
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 function)
DI = register number (multiple of 2 less than 0100h)
CX = word to write

**Return Registers:**
CF clear if successful
CF set on error
AH = status (00h,87h) (see Table 5-4)
EAX, EBX, ECX, and EDX may be modified
All other flags (except IF) may be modified

**Details:** This function may require up to 1024 bytes of stack; it will not enable interrupts if they were disabled before making the call. The meanings of BL and BH on entry may be exchanged in future implementations. **See also:** Functions B10Bh, B10Dh, and B18Ch

### INT 1Ah Function B10Dh
*Intel PCI BIOS v2.0c*

### *WRITE CONFIGURATION DWORD*

**Registers at call:**
AX = B10Dh
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 function)
DI = register number (multiple of 4 less than 0100h)
ECX = dword to write

**Return Registers:**
CF clear if successful
CF set on error
AH = status (00h,87h) (see Table 5-4)
EAX, EBX, ECX, and EDX may be modified
All other flags (except IF) may be modified

**Details:** This function may require up to 1024 bytes of stack; it will not enable interrupts if they were disabled before making the call. The meanings of BL and BH on entry may be exchanged in future implementations. **See also:** Functions B10Bh, B10Ch, and B18Dh

### INT 1Ah Function B181h
*Intel PCI BIOS v2.0c*

### *INSTALLATION CHECK (32-bit)*

**Registers at call:**
AX = B181h

**Return Registers:**
as for Function B101h

**See also:** Function B101h

### INT 1Ah Function B182h
*Intel PCI BIOS v2.0c*

### *FIND PCI DEVICE (32-bit)*

**Registers at call:**
AX = B182h
CX = device ID
DX = vendor ID
SI = device index (0-n)

**Return Registers:**
as for Function B102h

**See also:** Function B102h

### INT 1Ah Function B183h
*Intel PCI BIOS v2.0c*

### *FIND PCI CLASS CODE (32-bit)*

**Registers at call:**
AX = B183h
ECX = class code (bits 23-0)
SI = device index (0-n)

**Return Registers:**
as for Function B103h

---

### INT 1Ah Function B186h

*Intel PCI BIOS v2.0c*

***PCI BUS-SPECIFIC OPERATIONS (32-bit)***

**Registers at call:**
AX = B186h
BL = bus number
EDX = Special Cycle data

**Return Registers:**
as for Function B106h

**See also:** Function B106h

---

### INT 1Ah Function B188h

*Intel PCI BIOS v2.0c*

***READ CONFIGURATION BYTE (32-bit)***

**Registers at call:**
AX = B188h
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 function)
DI = register number (0000h-00FFh)

**Return Registers:**
as for Function B108h

**See also:** Function B108h

---

### INT 1Ah Function B189h

*Intel PCI BIOS v2.0c*

***READ CONFIGURATION WORD (32-bit)***

**Registers at call:**
AX = B189h
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 function)
DI = register number (0000h-00FFh)

**Return Registers:**
as for Function B109h

**See also:** Function B109h

---

### INT 1Ah Function B18Ah

*Intel PCI BIOS v2.0c*

***READ CONFIGURATION DWORD (32-bit)***

**Registers at call:**
AX = B18Ah
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 function)
DI = register number (0000h-00FFh)

**Return Registers:**
as for Function B10Ah

**See also:** Function B10Ah

---

### INT 1Ah Function B18Bh

*Intel PCI BIOS v2.0c*

***WRITE CONFIGURATION BYTE (32-bit)***

**Registers at call:**
AX = B18Bh
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 function)
DI = register number (0000h-00FFh)
CL = byte to write

**Return Registers:**
as for Function B10Bh

**See also:** Function B10Bh

---

### INT 1Ah Function B18Ch

*Intel PCI BIOS v2.0c*

***WRITE CONFIGURATION WORD (32-bit)***

**Registers at call:**
AX = B18Ch
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 function)
DI = register number (multiple of 2 less than 0100h)
CX = word to write

**Return Registers:**
as for Function B10Ch

**See also:** Function B10Ch

---

### INT 1Ah Function B18Dh

*Intel PCI BIOS v2.0c*

***WRITE CONFIGURATION DWORD (32-bit)***

**Registers at call:**
AX = B18Dh
BL = bus number
BH = device/function number (bits 7-3 device, bits 2-0 function)
DI = register number (multiple of 4 less than 0100h)
ECX = dword to write

**Return Registers:**
as for Function B10Dh

**See also:** Function B10Dh

## PCMCIA Socket Services

Originally designed to support memory cards, PCMCIA sockets now host a multitude of peripherals ranging from modems to hard disks, and are found on an increasing number of laptops, notebook computers, and even some desktop systems. PCMCIA Socket Services provide a hardware-independent interface for accessing varied peripherals.

---

### INT 1Ah Function 80h

*PCMCIA Socket Services*

***GET NUMBER OF ADAPTERS***

**Registers at call:**
AH = 80h

**Return Registers:**
CF clear if successful
CX = 5353h ('SS') if Socket Services installed
AL = number of adapters present (0-16)
AH destroyed
CF set on error
AH = error code (see Table 5-5)

**Conflicts:** PCjr sound (Chapter 22) **See also:** Function 83h

### 5-5 PCMCIA error codes:

| | |
|---|---|
| 01h | "BAD_ADAPTER" nonexistent adapter |
| 02h | "BAD_ATTRIBUTE" invalid attribute specified |
| 03h | "BAD_BASE" invalid system memory base address |
| 04h | "BAD_EDC" invalid EDC generator specified |
| 05h | "BAD_INDICATOR" invalid indicator specified |
| 06h | "BAD_IRQ" invalid IRQ channel specified |
| 07h | "BAD_OFFSET" invalid PCMCIA card offset specified |
| 08h | "BAD_PAGE" invalid page specified |
| 09h | "BAD_READ" unable to complete request |
| 0Ah | "BAD_SIZE" invalid window size specified |
| 0Bh | "BAD_SOCKET" nonexistent socket specified |
| 0Ch | "BAD_TECHNOLOGY" unsupported Card Technology for writes |
| 0Dh | "BAD_TYPE" unavailable window type specified |
| 0Eh | "BAD_VCC" invalid Vcc power level index specified |
| 0Fh | "BAD_VPP" invalid Vpp1 or Vpp2 power level index specified |
| 10h | "BAD_WAIT" invalid number of wait states specified |
| 11h | "BAD_WINDOW" nonexistent window specified |
| 12h | "BAD_WRITE" unable to complete request |
| 13h | "NO_ADAPTERS" no adapters installed, but Socket Services is present |
| 14h | "NO_CARD" no card in socket |

---

## INT 1Ah Function 81h

*PCMCIA Socket Services*

### *REGISTER STATUS CHANGE CALLBACK*

**Registers at call:**
AH = 81h
DS:DX -> callback routine (see Table 5-6) or 0000h:0000h to disable

**Return Registers:**
CF clear if successful
AH destroyed
CF set on error
AH = error code (see Table 5-5)

**Details:** The callback will be invoked on any socket changes whose notification has not been disabled with the status change enable mask; it may be invoked either while processing a hardware interrupt from the adapter or while processing the following Socket Services request. **Conflicts:** PCjr sound (Chapter 22) **See also:** Functions 80h and 82h

### 5-6 Status Change callback routine calling convention:

**Registers at call:**
AL = adapter number
BH = status change interrupt enable mask (see Table 5-7)
BL = socket number
DH = current socket status (see Table 5-8)
DL = current card status (see Table 5-9)

**Return Registers:**
all registers preserved

**Notes:** The callback may be invoked during a hardware interrupt, and may not call on Socket Services. The callback will be invoked once for each socket with a status change.

### 5-7 Bitfields for status change enable mask:

| | |
|---|---|
| bits 0,1 | reserved (0) |
| bit 2 | ejection request |
| bit 3 | insertion request |
| bit 4 | battery dead change |
| bit 5 | battery warning change |
| bit 6 | ready change |
| bit 7 | card detect change |

### 5-8 Bitfields for current socket status:

| | |
|---|---|
| bit 0 | reserved (0) |
| bit 1 | card locked |
| bit 2 | card ejection request pending |
| bit 3 | card insertion request pending |
| bit 4 | card ejection complete |
| bit 5 | card insertion complete |
| bit 6 | reserved (0) |
| bit 7 | card changed |

### 5-9 Bitfields for current card status:

| | |
|---|---|
| bit 0 | write protected |
| bits 1-3 | reserved (0) |
| bit 4 | battery voltage detect 1 (battery dead) |
| bit 5 | battery voltage detect 2 (battery warning) |
| bit 6 | ready |
| bit 7 | card detect |

---

## INT 1Ah Function 82h

*PCMCIA Socket Services*

### *REGISTER CARD TECHNOLOGY CALLBACK*

**Registers at call:**
AH = 82h
DS:DX -> callback routine (see Table 5-10) or 0000h:0000h

**Return Registers:**
CF clear if successful
  AH destroyed
CF set on error
  AH = error code (see Table 5-5)

**See also:** Functions 81h and 94h

### 5-10 Card Technology callback routine calling convention:

**Registers at call:**
ES:AX -> Low-Level Socket Services Routines (see Table 5-12)
BH = socket attributes (see Table 5-11)
CX = number of bytes or words to write
DS:SI -> data buffer to be written
DX:DI -> 26-bit linear card address
BP = card technology type

**Return Registers:**
CF clear if successful
CF set on error
  AH = error code (07h,0Ch,12h,14h) (see Table 5-5)

### 5-11 Bitfields for socket attributes:

| | |
|---|---|
| bit 0 | memory type (clear = common, set = attribute) |
| bit 1 | data width (clear = byte, set = word) |
| bit 2 | even bytes only (only valid if bit 1 set) |
| bit 3 | packed buffer |
| bits 4-7 | reserved (0) |

### 5-12 Format of Low-Level Socket Services Routines:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | offset of Write Many routine (see Table 5-13) |
| 02h | WORD | offset of Write One routine (see Table 5-14) |
| 04h | WORD | offset of Read One routine (see Table 5-15) |
| 06h | WORD | offset of Increment Offset routine (see Table 5-16) |
| 08h | WORD | offset of Set Offset routine (see Table 5-17) |
| 0Ah | WORD | offset of Get Status routine (see Table 5-18) |

### 5-13 Write Many routine calling convention:

**Registers at call:**
BH = socket attributes (see Table 5-11)
CX = number of bytes or words to write
DS:SI -> data to be written

**Return Registers:**
CF clear if successful
CF set on error

### 5-14 Write One routine calling convention:

**Registers at call:**
AL/AX = data to be written
BH = socket attributes (see Table 5-11)

**Return Registers:**
CF clear if successful
CF set on error

### 5-15 Read One routine calling convention:

**Registers at call:**
BH = socket attributes (see Table 5-11)

**Return Registers:**
CF clear if successful
  AL/AX = data read
CF set on error

### 5-16 Increment Offset routine calling convention:

**Registers at call:**
BH = socket attributes (see Table 5-11)

**Return Registers:**
CF clear if successful
CF set on error

### 5-17 Set Offset routine calling convention:

**Registers at call:**
DX:DI = new offset address

**Return Registers:**
CF clear if successful
CF set on error

### 5-18 Get Status routine calling convention:

**Register at call:**
none

**Return Registers:**
AL = current card status (see Table 5-9)

## INT 1Ah Function 83h
## *GET SOCKET SERVICES VERSION NUMBER*

*PCMCIA Socket Services*

**Registers at call:**
AH = 83h
AL = adapter number

**Return Registers:**
CF clear if successful
  AX = Socket Services version (BCD)
  BX = implementation version (BCD)
  CX = 5353h ("SS")
  DS:SI -> ASCIZ implementor description
CF set on error
  AH = error code (01h) (see Table 5-5)

**Details:** The current version (from the Revision A.00 documentation) of Socket Services is 1.00 (AX=0100h). **Conflicts:** PCjr sound (Chapter 22) **See also:** Function 80h

### INT 1Ah Function 84h

*PCMCIA Socket Services*

***INQUIRE ADAPTER***

**Registers at call:**
AH = 84h
AL = adapter number

**Return Registers:**
CF clear if successful
  AH destroyed
  BH = number of windows
  BL = number of sockets (1-16)
  CX = number of EDCs
  DH = capabilities (see Table 5-19)
  DL = status change interrupt used (only if DH bit 3 set)
CF set on error
  AH = error code (01h) (see Table 5-5)

**Conflicts:** PCjr sound (Chapter 22) **See also:** Functions 80h, 85h, and 87h

#### 5-19 Bitfields for capabilities:

| | |
|---|---|
| bit 0 | indicators are per-adapter rather than per-socket |
| bit 1 | power management is per-adapter rather than per-socket |
| bit 2 | data bus width is per-socket rather than per-window |
| bit 3 | status change interrupt |
| bit 4 | status change interrupt is software shareable |
| bit 5 | status change interrupt is hardware shareable |
| bits 6-7 | reserved (0) |

#### 5-20 Values for status change interrupt usage:

| | |
|---|---|
| 00h-0Fh | IRQ level |
| 10h | NMI |
| 11h | I/O check |
| 12h | bus error |
| 13h | vendor specific |
| 14h-FFh | reserved |

### INT 1Ah Function 85h

*PCMCIA Socket Services*

***GET ADAPTER***

**Registers at call:**
AH = 85h
AL = adapter number

**Return Registers:**
CF clear if successful
  AH destroyed
  DH = adapter attributes (see Table 5-21)
CF set on error
  AH = error code (01h) (see Table 5-5)

**Conflicts:** PCjr sound (Chapter 22) **See also:** Functions 84h and 86h

#### 5-21 Bitfields for adapter attributes:

| | |
|---|---|
| bit 0 | attempting to reduce power consumption |
| bit 1 | adapter preserves state information during reduced power consumption |
| bit 2 | enable status change interrupts |
| bit 3 | software share status change |
| bit 4 | hardware share status change |
| bits 5-7 | reserved (0) |

### INT 1Ah Function 86h

*PCMCIA Socket Services*

***SET ADAPTER***

**Registers at call:**
AH = 86h
AL = adapter number
DH = new adapter attributes (see Table 5-21)

**Return Registers:**
CF clear if successful
  AH destroyed
CF set on error
  AH = error code (01h) (see Table 5-5)

**See also:** Functions 84h and 85h

### INT 1Ah Function 87h

*PCMCIA Socket Services*

***INQUIRE WINDOW***

**Registers at call:**
AH = 87h
AL = adapter number
BH = window number

**Return Registers:**
CF clear if successful
  AH destroyed
  BL = capabilities (see Table 5-22)
  CX = bitmap of assignable sockets
  DH = EISA A15-A12 address lines (in bits 7-4, bits 3-0 = 0)
  DL = supported access speeds (see Table 5-23)
  DS:SI -> Memory Window Characteristics table (see Table 5-24)
  DS:DI -> I/O Window Characteristics table (see Table 5-25)
CF set on error
  AH = error code (01h,11h) (see Table 5-5)

#### 5-22 Bitfields for window capabilities:

| | |
|---|---|
| bit 0 | common memory |

| | |
|---|---|
| bit 1 | attribute memory |
| bit 2 | I/O space |
| bit 3 | EISA I/O mappable |
| bit 4 | separate enable for EISA comon space |
| bits 5-7 | reserved (0) |

**5-23 Values for supported access speeds:**

| | |
|---|---|
| bit 0 | WAIT line monitoring |
| bit 1 | 100 ns |
| bit 2 | 150 ns |
| bit 3 | 200 ns |
| bit 4 | 250 ns |
| bit 5 | 300 ns |
| bit 6 | 600 ns |
| bit 7 | reserved (0) |

**5-24 Format of Memory Window Characteristics table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | window capabilities (see Table 5-26) |
| 02h | WORD | minimum base address in 4K pages |
| 04h | WORD | maximum base address in 4K pages |
| 06h | WORD | minimum window size in 4K pages |
| 08h | WORD | maximum window size in 4K pages |
| 0Ah | WORD | window size granularity (4K units) |
| 0Ch | WORD | required base address alignment (4K units) |
| 0Eh | WORD | required card offset alignment (4K units) |

**5-25 Format of I/O Window Characteristics table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | window capabilities (see Table 5-26) |
| 02h | WORD | minimum base address in bytes |
| 04h | WORD | maximum base address in bytes |
| 06h | WORD | minimum window size in bytes |
| 08h | WORD | maximum window size in bytes |
| 0Ah | WORD | window size granularity (bytes) |

**5-26 Bitfields for window capabilities:**

| | |
|---|---|
| bit 0 | programmable base address |
| bit 1 | programmable window size |
| bit 2 | window disable/enable supported |
| bit 3 | 8-bit data bus |
| bit 4 | 16-bit data bus |
| bit 5 | base address alignment on size boundary required |
| bit 6 | power-of-two size granularity |
| ---memory windows--- | |
| bit 7 | card offset must be aligned on size boundary |
| bit 8 | paging hardware available |
| bit 9 | paging hardware shared |
| bit 10 | page disable/enable supported |
| bits 11-15 | reserved (0) |
| ---I/O windows--- | |
| bits 7-15 | reserved (0) |

## INT 1Ah Function 88h

### *GET WINDOW*

*PCMCIA Socket Services*

**Registers at call:**
AH = 88h
AL = adapter number
BH = window number

**Return Registers:**
CF clear if successful
AH destroyed
BL = socket number (0-16) (0 = not assigned)
CX = window size (bytes for I/O window, 4K units for memory window)
DH = window attributes (see Table 5-27)
DL = access speed (only one bit set) (see Table 5-23)
SI = window base address (bytes if I/O, 4K units if memory)
DI = card offset address (memory only, 4K units)
CF set on error
AH = error code (01h,11h) (see Table 5-5)

**See also:** Functions 87h, 89h, and 8Ah

**5-27 Bitfields for window attributes:**

| | |
|---|---|
| bit 0 | memory-mapped rather than I/O-mapped |
| bit 1 | attribute memory rather than common (memory-mapped)<br>EISA mapped (I/O) |
| bit 2 | enabled |

| | |
|---|---|
| bit 3 | 16-bit data path |
| bit 4 | subdivided into pages (memory-mapped only) |
| bit 5 | non-specific access slot enable (EISA-mapped only) |
| bits 6-7 | reserved (0) |

---

### INT 1Ah Function 89h

*PCMCIA Socket Services*

***SET WINDOW***

**Registers at call:**
AH = 89h
AL = adapter number
BH = window number
BL = socket number
CX = window size (bytes if I/O window,
4K units if memory window)
DH = window attributes (see Table 5-27)
DL = access speed (only one bit set) (see Table 5-23)
SI = window base address (bytes if I/O, 4K units if memory window)
DI = card offset addrress (memory only, 4K units)

**Return Registers:**
CF clear if successful
AH destroyed
CF set on error
AH = error code
(01h,03h,07h,08h,0Ah,0Bh,0Dh
10h,11h)
(see Table 5-5)

**See also:** Functions 87h, 88h, and 8Bh

---

### INT 1Ah Function 8Ah

*PCMCIA Socket Services*

***GET PAGE***

**Registers at call:**
AH = 8Ah
AL = adapter number
BH = window number
BL = page number

**Return Registers:**
CF clear if successful
AH destroyed
DX = page attributes
bit 0: page enabled
bits 15-1 reserved (0)
DI = memory card offset (4K units)
CF set on error
AH = error code (01h,08h,11h) (see Table 5-5)

**Details:** This function is only valid for memory-mapped windows. The socket being operated on is implied by the previous Function 89h call. **See also:** Functions 88h and 8Bh

---

### INT 1Ah Function 8Bh

*PCMCIA Socket Services*

***SET PAGE***

**Registers at call:**
AH = 8Bh
AL = adapter number
BH = window number
BL = page number
DX = page attributes
bit 0: page enabled
bits 15-1 reserved (0)
DI = memory card offset (4K units)

**Return Registers:**
CF clear if successful
AH destroyed
CF set on error
AH = error code (01h,02h,07h,08h,11h) (see Table 5-5)

**Details:** This function is only valid for memory-mapped windows. The socket being operated on is implied by the previous Function 89h call. **See also:** Functions 89h and 8Ah

---

### INT 1Ah Function 8Ch

*PCMCIA Socket Services*

***INQUIRE SOCKET***

**Registers at call:**
AH = 8Ch
AL = adapter number
BL = socket number (01h to
maximum supported by adapter)

**Return Registers:**
CF clear if successful
AH destroyed
DH = capabilities (see Table 5-28)
DL = hardware indicators (see Table 5-29)
DS:SI -> Socket Characteristics table (see Table 5-30)
DS:DI -> Power Management table (see Table 5-31)
CF set on error
AH = error code (01h,0Bh) (see Table 5-5)

**See also:** Functions 87h, 8Dh, and 8Eh

#### 5-28 Bitfields for socket capabilities:

| | |
|---|---|
| bit 0 | card change |
| bit 1 | card lock |
| bit 2 | insert card (motor control) |
| bit 3 | eject card (motor control) |
| bits 4-7 | reserved (0) |

#### 5-29 Bitfields for socket hardware indicators:

| | |
|---|---|
| bit 0 | busy status |
| bit 1 | write-protected |
| bit 2 | battery status |
| bit 3 | card lock status |

| | |
|---|---|
| bit 4 | XIP status (eXecute-In-Place) |
| bits 5-7 | reserved (0) |

### 5-30 Format of Socket Characteristics table:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | supported card types |
| | bit 0: memory card | |
| | bit 1: I/O card | |
| | bits 2-7 reserved (0) | |
| 02h | WORD | steerable IRQ levels (bit 0 = IRQ0 to bit 15 = IRQ15) |
| 04h | WORD | additional steerable IRQ levels |
| | bit 0: NMI | |
| | bit 1: I/O check | |
| | bit 2: bus error | |
| | bit 3: vendor-unique | |
| | bits 4-7 reserved (0) | |

### 5-31 Format of Power Management table:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of entries in table (0 if power management not avail) |
| 02h | 2N BYTEs | power levels |
| | byte 0 | voltage in 0.1V units |
| | byte 1 | power supply |
| | bit 7 | Vcc |
| | bit 6 | Vpp1 |
| | bit 5 | Vpp2 |

## INT 1Ah Function 8Dh

### *GET SOCKET*

*PCMCIA Socket Services*

**Registers at call:**

AH = 8Dh

AL = adapter number

BL = socket number (01h to maximum supported by adapter)

**Return Registers:**

CF clear if successful

AH destroyed

BH = status change interrupt enable mask (see Table 5-7)

CH = Vcc level (lower nybble)

CL = Vpp1 level (upper nybble) and Vpp2 level (lower nybble)

DH = current socket status (see Table 5-8)

DL = indicators (see Table 5-29)

SI = card type (see Table 5-32)

DI = IRQ level steering (I/O only) (see Table 5-33)

CF set on error

AH = error code (01h,0Bh) (see Table 5-5)

**See also:** Functions 8Ch and 8Eh

### 5-32 Bitfields for card type:

| | |
|---|---|
| bit 0 | memory |
| bit 1 | I/O |
| bits 2-15 | reserved (0) |

### 5-33 Bitfields for IRQ level steering:

| | |
|---|---|
| bits 0-4 | IRQ level (0-15=IRQ,16=NMI,17=I/O check,18=bus error,19=vendor) |
| bits 5-14 | reserved (0) |
| bit 15 | interrupt steering enabled |

## INT 1Ah Function 8Eh

### *SET SOCKET*

*PCMCIA Socket Services*

**Registers at call:**

AH = 8Eh

AL = adapter number

BL = socket number (01h to maximum supported by adapter)

BH = status change interrupt enable mask (see Table 5-7)

CL = Vpp1 level (upper nybble) and Vpp2 level (lower nybble)

DH = current socket status (see Table 5-8)

DL = indicators (see Table 5-29)

SI = card type (see Table 5-32)

DI = IRQ level steering (I/O only) (see Table 5-33)

**Return Registers:**

CF clear if successful

AH destroyed

CF set on error

AH = error code (01h,02h,05h,06h,0Bh,0Eh, 0Fh) (see Table 5-5)

**See also:** Functions 8Ch and 8Dh

## INT 1Ah Function 8Fh

### *GET CARD*

*PCMCIA Socket Services*

**Registers at call:**

AH = 8Fh

AL = adapter number

BL = socket number (01h to maximum supported by adapter)

**Return Registers:**

CF clear if successful

AH destroyed

CF set on error

AH = error code (01h,0Bh,14h) (see Table 5-5)

### INT 1Ah Function 90h

*PCMCIA Socket Services*

***RESET CARD***

**Registers at call:**
AH = 90h
AL = adapter number
BL = socket number (01h to maximum supported by adapter)

**Return Registers:**
CF clear if successful
  AH destroyed
CF set on error
  AH = error code (01h,0Bh,14h) (see Table 5-5)

**Details:** This function toggles RESET pin of the specified card, but does not wait after toggling the pin; it is the caller's responsibility to avoid accessing the card before it is ready again.

### INT 1Ah Function 91h

*PCMCIA Socket Services*

***READ ONE***

**Registers at call:**
AH = 91h
AL = adapter number
BL = socket number (01h to maximum supported by adapter)
BH = attributes (see Table 5-34)
DX:SI = card address

**Return Registers:**
CF clear if successful
  AH destroyed
  CL/CX = value read
CF set on error
  AH = error code (01h,07h,09h,0Bh,14h) (see Table 5-5)
  CX may be destroyed

**Details:** This function is only valid for I/O-mapped sockets. **See also:** Functions 92h and 93h, MS-DOS INT 21h Function 440Dh (Chapter 24)

**5-34 Bitfields for attributes:**

| | |
|---|---|
| bit 0 | attribute memory instead of common memory |
| bit 1 | word rather than byte |
| bit 2 | even bytes only |

### INT 1Ah Function 92h

*PCMCIA Socket Services*

***WRITE ONE***

**Registers at call:**
AH = 92h
AL = adapter number
BL = socket number (01h to maximum supported by adapter)
BH = attributes (see Table 5-34)
CL/CX = value to write
DX:SI = card address

**Return Registers:**
CF clear if successful
  AH destroyed
CF set on error
  AH = error code (01h,07h,0Bh,12h,14h) (see Table 5-5)

**Details:** This function is only valid for I/O-mapped sockets; it also does not implement Card Technology handling--use Function 94h when writing to non-RAM technologies. **See also:** Functions 91h and 94h, MS-DOS INT 21h Function 440Dh (Chapter 24)

### INT 1Ah Function 93h

*PCMCIA Socket Services*

***READ MULTIPLE***

**Registers at call:**
AH = 93h
AL = adapter number
BL = socket number (01h to maximum supported by adapter)
BH = attributes (see Table 5-34)
CX = number of bytes or words to read
DX:SI = card address
DS:DI -> data buffer to be filled

**Return Registers:**
CF clear if successful
  AH destroyed
CF set on error
  AH = error code (01h,07h,09h,0Bh,14h) (see Table 5-5)

**Details:** This function is only available on I/O-mapped sockets. **See also:** Functions 91h and 94h, MS-DOS INT 21h Function 440Dh (Chapter 24)

### INT 1Ah Function 94h

*PCMCIA Socket Services*

***WRITE MULTIPLE***

**Registers at call:**
AH = 94h
AL = adapter number
BL = socket number (01h to maximum supported by adapter)
BH = attributes (see Table 5-34)
CX = number of bytes or words to read
DX:DI = card address
DS:SI -> buffer containing data
BP = Card Technology type (0000h = RAM)

**Return Registers:**
CF clear if successful
  AH destroyed
CF set on error
  AH = error code (01h,07h,0Bh,0Ch,12h,14h) (see Table 5-5)

**Details:** This function is available only on I/O-mapped sockets. Socket Services calls the Card Technology callback (see Table 5-10) for any card technology it does not directly support. **See also:** Functions 92h and 93h, MS-DOS INT 21h Function 440Dh (Chapter 24)

### INT 1Ah Function 95h

*PCMCIA Socket Services*

***INQUIRE ERROR DETECTION CODE***

**Registers at call:**
AH = 95h
AL = adapter number
BH = EDC generator number

**Return Registers:**
CF clear if successful
  AH destroyed
  CX = bitmap of assignable sockets
  DH = EDC capabilities (see Table 5-35)
  DL = supported EDC types (see Table 5-36)
CF set on error
  AH = error code (01h,04h) (see Table 5-5)

**See also:** Functions 96h and 9Ch

### 5-35 Bitfields for EDC capabilities:

| | |
|---|---|
| bit 0 | unidirectional only generation |
| bit 1 | bidirectional only generation |
| bit 2 | register-based (I/O-mapped) support |
| bit 3 | memory-mapped support |
| bit 4 | pausable |
| bits 5-7 | reserved (0) |

### 5-36 Bitfields for supported EDC types:

| | |
|---|---|
| bit 0 | 8-bit checksum |
| bit 1 | 16-bit CRC-SDLC |
| bits 2-7 r | eserved (0) |

---

## INT 1Ah Function 96h

*PCMCIA Socket Services*

### *GET ERROR DETECTION CODE*

**Registers at call:**
AH = 96h
AL = adapter number
BH = EDC generator number

**Return Registers:**
CF clear if successful
  AH destroyed
  BL = socket number
  DH = EDC attributes (see Table 5-37)
  DL = EDC type (see Table 5-36) (only one bit set)
CF set on error
  AH = error code (01h,04h) (see Table 5-5)

**See also:** Functions 95h, 97h, and 9Ch

### 5-37 Bitfields for EDC attributes:

| | |
|---|---|
| bit 0 | unidirectional only |
| bit 1 | (if bit 0 set) clear=read, set=write |
| bits 2-7 | reserved (0) |

---

## INT 1Ah Function 97h

*PCMCIA Socket Services*

### *SET ERROR DETECTION CODE*

**Registers at call:**
AH = 97h
AL = adapter number
BH = EDC generator
BL = socket number
DH = EDC attributes (see Table 5-37)
DL = EDC type (see Table 5-36) (only one bit may be set)

**Return Registers:**
CF clear if successful
  AH destroyed
CF set on error
  AH = error code (01h,02h,04h,0Bh)
  (see Table 5-5)

**See also:** Functions 96h and 9Ch

---

## INT 1Ah Functions 98h to 9Bh

*PCMCIA Socket Services*

### *START/STOP/PAUSE/RESUME ERROR DETECTION CODE*

**Registers at call:**
AH = function
  98h start error detection code
  99h pause EDC
  9Ah resume EDC
  9Bh stop EDC
AL = adapter number
BH = EDC generator

**Return Registers:**
CF clear if successful
  AH destroyed
CF set on error
  AH = error code (01h,04h) (see Table 5-5)

**See also:** Functions 96h and 9Ch

---

## INT 1Ah Function 9Ch

*PCMCIA Socket Services*

### *READ ERROR DETECTION CODE*

**Registers at call:**
AH = 9Ch
AL = adapter number
BH = EDC generator

**Return Registers:**
CF clear if successful
  AH destroyed
  DL/DX = computed checksum or CRC
CF set on error
  AH = error code (01h,04h) (see Table 5-5)

**See also:** Functions 95h, 96h, 98h, 99h, and 9BhINT 21h Function 440Dh

CHAPTER 6

# *Japanese ROM BIOS Extensions*

## AX PC (Japanese)

The AX PC is the Japanese version of the IBM PC/AT.

---

### INT 10h Function 5000h
*AX PC VIDEO*

***SET SCREEN COUNTRY CODE***

**Registers at call:**
AX = 5000h
BX = country code
    0001h USA (English), 0051h Japan

**Return Registers:**
AL = status
    00h successful
    01h bad country code
    02h other error

**See also:** Function 5001h, INT 16h Function 5000h

---

### INT 10h Function 5001h
*AX PC VIDEO*

***GET SCREEN COUNTRY CODE***

**Registers at call:**
AX = 5001h

**Return Registers:**
AL = status
    00h successful
        BX = country code
    02h error

**See also:** Function 5000h, INT 16h Function 5001h,BIOS Video INT 10h Function 00h (Chapter 8), MSDOS INT 21h Function 38h (Chapter 24)

---

### INT 10h Function 5100h
*AX PC VIDEO*

***REGISTER EXTERNAL CHARACTER***

**Purpose:** Define the screen appearance of one of the two-byte character codes.

**Registers at call:**
AX = 5100h
BH = character width in bits (10h)
BL = character height (10h)
DX = character code (DH = F0h-F3h, DL=40h-7Eh,80h-FCh)
ES:BP -> character bitmap

**Return Registers:**
AL = status (00h successful, 01h failed)

**Conflicts:** SCROLOCK (Chapter 8)

**See also:** Function 5101h, BIOS INT 1Fh (Chapter 8)

---

### INT 10h Function 5101h
*AX PC VIDEO*

***READ CHARACTER***

**Purpose:** Determine the screen appearance of the specified character.

**Registers at call:**
AX = 5101h
BH = character width in bits
BL = character height
DX = character code (DH = 00h if 8-bit character)
ES:BP -> buffer for character bitmap

**Return Registers:**
AL = status (00h successful, 01h failed)

**Conflicts:** SCROLOCK (Chapter 8)
**See also:** Function 5100h, BIOS Function 09h (Chapter 8)

---

### INT 10h Function 5200h
### *SET VIRTUAL TEXT RAM BUFFER*

*AX PC VIDEO*

**Registers at call:**
AX = 5200h
BX = segment of buffer
**See also:** Function 5201h

---

### INT 10h Function 5201h
### *GET VIRTUAL TEXT RAM BUFFER*

*AX PC VIDEO*

**Registers at call:**
AX = 5201h

**Return Registers:**
BX = segment of buffer or 0000h if failed

**See also:** Functions 5200h and 8300h, TopView Function FEh (Chapter 44)

---

### INT 10h Function 8200h
### *GET/SET SCROLL MODE*

*AX PC VIDEO*

**Registers at call:**
AX = 8200h
BL = new scroll mode (00h dynamic, 01h software) or FFh to get current mode

**Return Registers:**
AL = scroll mode (current mode if BL=FFh, previous mode otherwise)

**Conflicts:** DESQview (Chapter 44)
**See also:** BIOS Functions 06h and 07h (Chapter 8)

---

### INT 10h Function 8300h
### *GET VIDEO RAM ADDRESS*

*AX PC VIDEO*

**Registers at call:**
AX = 8300h

**Return Registers:**
AX = offset of video RAM
ES:BX -> virtual text RAM buffer

**See also:** Function 5201h

---

### INT 16h Function 5000h
### *SET KEYBOARD COUNTRY CODE*

*AX PC KEYBOARD*

**Registers at call:**
AX = 5000h
BX = country code
    0001h USA (English), 0051h Japan

**Return Registers:**
AL = status
    00h successful
    01h bad country code
    02h other error

**See also:** Function 5001h, INT 10h Function 5000h, INT 17h Function 5000h

---

### INT 16h Function 5001h
### *GET KEYBOARD COUNTRY CODE*

*AX PC KEYBOARD*

**Registers at call:**
AX = 5001h

**Return Registers:**
AL = status
    00h successful

BX = country code
02h error

**See also:** Function 5000h, INT 10h Function 5001h, INT 17h Function 5001h

---

### INT 16h Function 51h
*AX PC KEYBOARD*

***READ SHIFT KEY STATUS***

**Registers at call:**
AH = 51h

**Return Registers:**
AL = standard shift key states (see Function 12h in Chapter 4)
AH = Kana lock (00h off, 01h on)

**See also:** BIOS Functions 02h, 12h, and 22h (Chapter 4)

---

### INT 17h Function 5000h
*AX (Japanese AT) PRINTER*

***SET PRINTER COUNTRY CODE***

**Registers at call:**
AX = 5000h
BX = country code
0001h USA (English), 0051h Japan

**Return Registers:**
AL = status
00h successful
01h bad country code
02h other error

**See also:** Functions 5001h and 51h, INT 10h Function 5000h, INT 16h Function 5000h

---

### INT 17h Function 5001h
*AX (Japanese AT) PRINTER*

***GET PRINTER COUNTRY CODE***

**Registers at call:**
AX = 5001h

**Return Registers:**
AL = status
00h successful
BX = country code
02h error

**See also:** Functions 5000h and 51h, INT 10h Function 5001h, INT 16h Function 5001h

---

### INT 17h Function 51h
*AX (Japanese AT) PRINTER*

***JIS to Shift-JIS CONVERSION***

**Registers at call:**
AH = 51h
DX = 2-byte JIS code

**Return Registers:**
DX = shift-JIS value or 0000h on error

**See also:** Functions 5000h, 52h, and 84h

---

### INT 17h Function 52h
*AX (Japanese AT) PRINTER*

***Shift-JIS to JIS CONVERSION***

**Registers at call:**
AH = 52h
DX = 2-byte shift-JIS code

**Return Registers:**
DX = JIS code or 0000h on error

**See also:** Functions 51h and 84h

---

### INT 17h Function 84h
*AX (Japanese AT) PRINTER*

***OUTPUT CHARACTER WITHOUT CONVERSION***

**Registers at call:**
AH = 84h
AL = character
DX = printer number

**Return Registers:**
AH = printer status (see BIOS Function 00h In Chapter 15)

**Conflicts:** Alloy NTNX (NI)

**See also:** Function 85h, BIOS Function 00h (Chapter 15)

### INT 17h Function 85h
### *ENABLE/DISABLE CHARACTER CONVERSION*

*AX (Japanese AT) PRINTER*

**Registers at call:**
AH = 85h
AL = new state (00h enabled, 01h disabled)

**Return Registers:**
none

**See also:** Function 84h

## NEC

The NEC PC-9800 series of computers are Japanese MSDOS-compatibles (not IBM-compatibles) which almost totally dominated the Japanese market in personal computers for several years.

### INT 18h Function 00h
### *GET KEYSTROKE*

*NEC PC-9800 series KEYBOARD*

**Registers at call:**
AH = 00h

**Return Registers:**
AX = keystroke

**See also:** Functions 01h and 02h, BIOS INT 16h Function 00h (Chapter 4)

### INT 18h Function 01h
### *CHECK FOR KEYSTROKE*

*NEC PC-9800 series KEYBOARD*

**Registers at call:**
AH = 01h

**Return Registers:**
ZF set if no keystroke available
ZF clear if keystroke available
    AX = keystroke

**See also:** Functions 00h and 02h, BIOS INT 16h Function 01h (Chapter 4)

### INT 18h Function 02h
### *GET SHIFT STATUS*

*NEC PC-9800 series KEYBOARD*

**Registers at call:**
AH = 02h

**Return Registers:**
AL = shift flags

**See also:** Functions 00h, 02h, and 03h, BIOS INT 16h Function 02h (Chapter 4)

### INT 18h Function 03h
### *INITIALIZE*

*NEC PC-9800 series KEYBOARD*

**Registers at call:**
AH = 03h
*other registers, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Functions 00h and 04h

### INT 18h Function 04h
### *KEY PRESSED*

*NEC PC-9800 series KEYBOARD*

**Registers at call:**
AH = 04h
*other registers, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Details are not available at this time.
**See also:** Functions 00h and 02h, BIOS INT 16h Functions 00h to 02h (Chapter 4)

### INT 18h
### *MULTIPLE FUNCTIONS*

*NEC PC-9800 series VIDEO*

**Registers at call:**
AH = function
    0Ah set video mode
    0Bh get video mode

**Return Registers:**
*unknown*

0Ch start text screen display
0Dh end text screen display
0Eh set single display area
0Fh set multiple display area
10h set cursor shape
11h display cursor
12h terminate cursor
13h set cursor position
14h read font patter
16h initialize text video RAM
1Ah define user character
*other registers vary by function*

**Details:** Details of these functions are not available at this time.

Text video RAM is located at segments A000h (characters) and A200h (attributes).

---

### INT 1Ah Function 10h
*NEC PC-9800 series PRINTER*

### *INITIALIZE*

**Registers at call:**
AH = 10h
*other registers, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Functions 11h and 12h, BIOS INT 17h Function 01h (Chapter 15)

---

### INT 1Ah Function 11h
*NEC PC-9800 series PRINTER*

### *OUTPUT CHARACTER*

**Registers at call:**
AH = 11h
*other registers, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Functions 10h and 12h, BIOS INT 17h Function 00h (Chapter 15)

---

### INT 1Ah Function 12h
*NEC PC-9800 series PRINTER*

### *SENSE STATUS*

**Registers at call:**
AH = 12h
*other registers, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Functions 10h and 11h, BIOS INT 17h Function 02h (Chapter 15)

## CHAPTER 7

# Vendor-Specific ROM BIOS Extensions

## Acorn BBC Master 512

The Acorn BBC Master 512 is an 80186-based add-on board for the 6502-based Master 128 which uses the original CPU as an I/O processor.

---

### INT 40h

*Acorn BBC Master 512*

***"OSFIND" - OPEN FILE***

**Registers at call:**
AL = operation
  00h close file
  40h open file for reading
  80h open file for writing
  C0h open file for random access
DS:BX -> CR-terminated filename

**Return Registers:**
AL = file handle (00h if file closed or could not be opened)

**Conflicts:** Z100 (below), BIOS Diskette (Chapter 10)
**See also:** INT 41h, INT 42h, INT 43h, INT 44h, INT 4Ch

---

### INT 41h

*Acorn BBC Master 512*

***"OSGBPB" - MULTI-BYTE GET/PUT***

**Registers at call:**
AL = function
  01h put bytes sequentially
  02h put bytes, ignoring sequential pointer
  03h get bytes sequentially
  04h get bytes, ignoring sequential pointer
  05h get media title and boot option
  06h get current device and directory
  07h get current library and device
  08h search directory
DS:BX -> control block (see Table 7-1)

**Return Registers:**
CF clear if successful
CF set on error
AL = 00h if operation attempted
AL unchanged if unsupported function

**Conflicts:** Z100 (below), BIOS Data (Chapter 10), MS Windows (Chapter 57)
**See also:** INT 40h, INT 42h, INT 43h

**7-1 Format of control block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | file handle |
| 01h | DWORD | pointer to data in either I/O processor or Tube processor |
| 05h | DWORD | number of bytes to be transferred |
| 09h | DWORD | transfer address |

---

**INT 42h** *Acorn BBS Master 512*

***"OSBPUT" - WRITE SINGLE BYTE TO FILE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AL = byte to be written | flags destroyed |
| BH = file handle | |

**Conflicts:** Z100 (below), BIOS Video (Chapter 8), Western Digital SuperBIOS (Chapter 10)

**See also:** INT 40h, INT 41h, INT 43h, INT 47h, INT 49h

---

**INT 43h** *Acorn BBC Master 512*

***"OSBGET" - READ SINGLE BYTE FROM FILE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| BH = file handle | CF clear if successful<br>AL = byte read from file<br>CF set on error |

**Conflicts:** Z100 (below), BIOS Video (Chapter 8)

**See also:** INT 40h, INT 41h, INT 42h, INT 46h

---

**INT 44h Function 00h** *Acorn BBC Master 512*

***"OSARGS" - GET/SET FILE PARAMS FOR OPEN FILE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 00h | |
| AL = function | |
| 00h get current filing system | AL = filing system (see Table 7-2) |
| 01h get address of commandline tail<br>BX -> 4-byte data buffer | BX buffer filled with address of command tail in I/O processor address space (use INT 4Ah Function 05h to retrieve) |
| FFh flush all files onto secondary storage | |

**Details:** The commandline tail is terminated with a carriage return (0Dh).

**Conflicts:** Z100 (below), BIOS Video (Chapter 8), Virus (Chapter 59), NetWare (NI), IBM 3270-PC (NI)

**See also:** INT 40h, INT 45h

**7-2 Values for filing system:**

| | |
|---|---|
| 00h | none |
| 01h | 1200 bps cassette |
| 02h | 300 bps cassette |
| 03h | ROM FS |
| 04h | DFS |
| 05h | ANFS/NFS |
| 06h | TFS |
| 08h | ADFS |

---

**INT 44h** *Acorn BBC Master 512*

***"OSARGS" - GET/SET FILE PARAMS FOR OPEN FILE***

**Registers at call:**

AH = nonzero file handle

AL = function
    00h get sequential pointer for file
    01h set sequential pointer for file
    02h get length of file
BX -> 4-byte data buffer

**Return Registers:**
BX buffer updated if appropriate

**Conflicts:** Z100 (below), BIOS Video (Chapter 8), Virus (Chapter 59), NetWare (NI), IBM 3270-PC (NI)
**See also:** INT 40h, INT 41h, INT 44h Function 00h, INT 45h, INT 4Ah

---

### INT 45h
*Acorn BBC Master 512*

### *"OSFILE" - READ/WRITE FILE OR DIRECTORY INFO*

**Registers at call:**
AL = function
    00h save block of memory as file
    01h update directory entry for existing file
    02h set load address for existing file
    03h set execution address for existing file
    04h set attributes for existing file
    05h read directory
    06h delete file
    FFh load file
DS:BX -> control block (see Table 7-3)

**Return Registers:**
FLAGS destroyed
AL = file type
    00h not found
    01h file found
    02h directory found
    FFh protected file

**Conflicts:** Z100 (below)
**See also:** INT 40h, INT 41h, INT 44h, INT 46h

**7-3 Format of control block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | address of CR-terminated filename |
| 02h | DWORD | load address of file |
| 06h | DWORD | execution address of file |
| 0Ah | DWORD | start address of data to save |
| 0Eh | DWORD | end address of data to save, or file attributes<br>file attributes in low byte (see Table 7-4), other three bytes are filing-system specific file attributes |

**7-4 Bitfields for file attributes:**

| | |
|---|---|
| bit 0 | no owner read access |
| bit 1 | no owner write access |
| bit 2 | not executable by owner |
| bit 3 | not deletable by owner |
| bit 4 | no public read access |
| bit 5 | no public write access |
| bit 6 | not executable with public access |
| bit 7 | not deletable with public access |

---

### INT 46h
*Acorn BBC Master 512*

### *"OSRDCH" - GET CHARACTER FROM CURRENT INPUT STREAM*

**Registers at call:**
none

**Return Registers:**
CF clear if successful
    AL = character read
CF set on error
    AL = error code

**Conflicts:** Z100 (below), BIOS Data (Chapter 10)
**See also:** INT 40h, INT 43h, INT 47h, INT 49h

### INT 47h

*Acorn BBC Master 512*

### "OSWRCH" - WRITE CHARACTER TO CURRENT OUTPUT STREAM

| Registers at call: | Return Registers: |
|---|---|
| AL = character to be written | FLAGS destroyed |

**Conflicts:** Z100 (below), Western Digital SuperBIOS (Chapter 10), SQL Base (Chapter 54)
**See also:** INT 40h, INT 46h, INT 49h

### INT 48h

*Acorn BBC Master 512*

### "OSNEWL" - SEND NEWLINE TO OUTPUT STREAM

| Registers at call: | Return Registers: |
|---|---|
| none | FLAGS destroyed |

**Details:** Writes a carriage return (0Dh) followed by a linefeed (0Ah).
**Conflicts:** Z100 (below), Western Digital SuperBIOS (Chapter 10), PCjr (Chapter 13), Compaq UILIB (Chapter 54), Watstar PC Network (NI)
**See also:** INT 40h, INT 47h, INT 49h

### INT 49h

*Acorn BBC Master 512*

### "OSASCI" - WRITE CHARACTER TO CURRENT OUTPUT STREAM

| Registers at call: | Return Registers: |
|---|---|
| AL = character to be written | FLAGS destroyed |

**Details:** Converts carriage return (0Dh) into CRLF sequence (0Dh 0Ah).
**Conflicts:** Z100 (below), Texas Instruments (Chapter 8), PCjr (Chapter 13), MAGic (Chapter 23), Watstar PC Network (NI)
**See also:** INT 40h, INT 46h, INT 47h, INT 48h

### INT 4Ah

*Acorn BBC Master 512*

### "OSWORD" - MISC FUNCTIONS USING CONTROL BLOCK

| Registers at call: | Return Registers: |
|---|---|
| AL = function code<br>FAh transfer data between 80186 and 65C12 I/O processor<br>DS:BX -> control block (see Table 7-5) | FLAGS destroyed<br>control block updated. |

**Details:** There are more functions than are listed here, but details are not available.
**Conflicts:** Z100 (below), User Alarm (Chapter 4)
**See also:** INT 40h, INT 4Bh, INT 4Ch

**7-5 Format of control block for function FAh:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | number of parameters sent to I/O processor (0Dh,0Eh) |
| 01h | BYTE | number of parameters read from I/O processor (01h) |
| 02h | DWORD | I/O processor address |
| 06h | DWORD | 80186 segment:offset address |
| 0Ah | WORD | number of bytes to transfer |
| 0Ch | BYTE | operation type<br>00h write to 65C12 at 24 us/byte<br>01h read from 65C12 at 24 us/byte<br>02h write to 65C12 at 26 us/word<br>03h read from 65C12 at 26 us/word<br>04h write to 65C12 at 10 us/byte using 256-byte blocks<br>05h read from 65C12 at 10 us/byte using 256-byte blocks |
| 0Dh | BYTE | 65C12 memory access control (only used if offset 00h = 0Eh)<br>bit 7: unused |

bit 6: always use main screen memory if I/O addr 3000h-7FFFh (overrides bit 5)
bit 5: use shadow screen memory if screen address specified
bit 4: use current ROM rather than ROM selected by bits 3-0 (only if I/O address between 8000h and BFFFh)
bits 3-0: paged ROM number

---

### INT 4Bh

*Acorn BBC Master 512*

### *"OSBYTE" - MISC FUNCTIONS USING REGISTER PARAMS*

**Registers at call:**
AL = function code
BL = first parameter
BH = second parameter (if needed)

**Return Registers:**
BL = first return parameter
BH = second return parameter
CF depends on function

**Conflicts:** Z100 (below), CAM SCSI (Chapter 11), IBM SCSI (Chapter 11), VDS (Chapter 18)
**See also:** INT 40h, INT 4Ah, INT 4Ch

---

### INT 4Ch

*Acorn BBC Master 512*

### *"OSCLI" - INTERPRET COMMAND LINE*

**Registers at call:**
DS:BX -> CR-terminated command string

**Return Registers:**
FLAGS destroyed

**Conflicts:** Z100 (below)
**See also:** INT 40h, INT 4Ah, INT 4Bh

## Amstrad

The Amstrad PC1512 is a largely IBM-compatible machine which was popular primarily in Europe.

---

### INT 15h Function 01h

*Amstrad PC1512*

### *WRITE DATA TO NON-VOLATILE RAM*

**Registers at call:**
AH = 01h
AL = NVRAM location (00h to 3Fh)
BL = NVRAM data value

**Return Registers:**
AH = return code
    00h OK
    01h address bad
    02h write error

**Conflicts:** Cassette (Chapter 4), MultiDOS Plus (Chapter 45), VMiX (Chapter 46)
**See also:** Function 02h

**7-6 Format of NVRAM:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | time of day: seconds |
| 01h | BYTE | alarm time: seconds |
| 02h | BYTE | time of day: minutes |
| 03h | BYTE | alarm time: minutes |
| 04h | BYTE | time of day: hours |
| 05h | BYTE | alarm time: hours |
| 06h | BYTE | day of week, 1 = Sunday |
| 07h | BYTE | day of month |
| 08h | BYTE | month |
| 09h | BYTE | year mod 100 |
| 0Ah | BYTE | RTC status register A<br>bit 7: set if date/time being updated<br>bits 6-4: time base speed, default 010 = 32768 Hz |

| | | |
|---|---|---|
| | | bits 3-0: interrupt rate selection, default 0110 = 1024 Hz |
| 0Bh | BYTE | RTC status register B (see Table 7-7) |
| 0Ch | BYTE | RTC status register C (read-only)<br>bit 7: IRQF flag<br>bit 6: PF flag<br>bit 5: AF flag<br>bit 4: UF flag |
| 0Dh | BYTE | RTC status register D<br>bit 7: battery good |
| 0Eh | 6 BYTEs | time and date machine last used |
| 14h | BYTE | user RAM checksum |
| 15h | WORD | Enter key scancode/ASCII code |
| 17h | WORD | Forward delete key scancode/ASCII code |
| 19h | WORD | Joystick fire button 1 scancode/ASCII code |
| 1Bh | WORD | Joystick fire button 2 scancode/ASCII code |
| 1Dh | WORD | mouse button 1 scancode/ASCII code |
| 1Fh | WORD | mouse button 2 scancode/ASCII code |
| 21h | BYTE | mouse X scaling factor |
| 22h | BYTE | mouse Y scaling factor |
| 23h | BYTE | initial VDU mode and drive count |
| 24h | BYTE | initial VDU character attribute |
| 25h | BYTE | size of RAM disk in 2K blocks |
| 26h | BYTE | initial system UART setup byte |
| 27h | BYTE | initial external UART setup byte |
| 28h | 24 BYTEs | available for user application |

**Details:** Bytes 00h-0Dh are the same on the IBM AT as they are used/updated by the clock chip.

**7-7 Bitfields For RTC status register B:**

| | |
|---|---|
| bit 7 | clear if normal update, set if abort update |
| bit 6 | periodic interrupt enable |
| bit 5 | alarm interrupt enable |
| bit 4 | update end interrupt enable |
| bit 3 | square wave enable |
| bit 2 | date mode (clear = BCD, set = binary) |
| bit 1 | 24-hour format |
| bit 0 | daylight saving time enable. |

---

## INT 15h Function 02h

*Amstrad PC1512*

### *READ DATA FROM NON-VOLATILE RAM*

**Registers at call:**
AH = 02h
AL = NVRAM location (00h to 3Fh)

**Return Registers:**
AH = return code
00h OK
01h address bad
02h checksum error
AL = NVRAM data value

**Conflicts:** Cassette (Chapter 4), MultiDOS Plus (Chapter 45), VMiX (Chapter 46)
**See also:** Function 01h

---

**INT 15h Function 03h** *Amstrad PC1512*

***WRITE VDU COLOR PLANE WRITE REGISTER***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 03h | none |
| AL = value (I,R,G,B bits) | |

**Conflicts:** Cassette (Chapter 4), MultiDOS Plus (Chapter 45), VMiX (Chapter 46)
**See also:** Function 04h

---

**INT 15h Function 04h** *Amstrad PC1512*

***WRITE VDU COLOR PLANE READ REGISTER***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 04h | none |
| AL = value (RDSEL1 and RDSEL0) | |

**Conflicts:** ABIOS (Chapter 4), MultiDOS Plus (Chapter 45), VMiX (Chapter 46)
**See also:** Functions 03h and 05h

---

**INT 15h Function 05h** *Amstrad PC1512*

***WRITE VDU GRAPHICS BORDER REGISTER***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 05h | none |
| AL = value (I,R,G,B bits) | |

**Conflicts:** ABIOS (Chapter 4), MultiDOS Plus (Chapter 45), VMiX (Chapter 46)
**See also:** Function 04h

---

**INT 15h Function 06h** *Amstrad PC1512*

***GET ROS VERSION NUMBER***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 06h | BX = version number |

**Conflicts:** MultiDOS Plus (Chapter 45), VMiX (Chapter 46)

## Atari

The Atari Portfolio is a palmtop computer.

---

**INT 60h** *Atari Portfolio*

***USER INTERFACE FUNCTIONS***

**Purpose:** This interrupt supplies a number of subfunctions which perform such functions as drawing boxes and menus, and provide input line editing.
**Conflicts:** see Table 2-3
**See also:** INT 61h

---

**INT 61h** *Atari Portfolio*

***EXTENDED BIOS***

**Purpose:** This interrupt provides subfunctions such as turning off the machine, accessing internal variables, and mapping memory cards.
**Conflicts:** see Table 2-3
**See also:** INT 60h

## AT&T

The AT&T 6300 was an early IBM-compatible manufactured by Olivetti.

---

### INT 1Ah Function FEh
### *READ TIME AND DATE*

*AT&T 6300*

**Registers at call:**
AH = FEh

**Return Registers:**
BX = day count (0 = Jan 1, 1984)
CH = hour
CL = minute
DH = second
DL = hundredths

**See also:** Function FFh, DATEFIX Function CCCCh (Chapter 66), DOS INT 21h Functions 2Ah and 2Ch (Chapter 24)

---

### INT 1Ah Function FFh
### *SET TIME AND DATE*

*AT&T 6300*

**Registers at call:**
AH = FFh
BX = day count (0 = Jan 1, 1984)
CH = hour
CL = minute
DH = second
DL = hundredths

**Return Registers:**
*unknown*

**See also:** Function FEh, DATEFIX Function CCCCh (Chapter 66), DOS INT 21h Functions 2Bh and 2Dh (Chapter 24)

## Compaq

---

### INT 15h Function 4000h
### *READ LCD/PLASMA TIMEOUT*

*Compaq SLT/286 or Portable 386*

**Registers at call:**
AX = 4000h

**Return Registers:**
AX = 4000h
CL = 00h timeout disabled else timeout in minutes

**Conflicts:** IBM Convertible (Chapter 4)
**See also:** Functions 4001h and 4600h

---

### INT 15h Function 4001h
### *SET LCD/PLASMA TIMEOUT*

*Compaq SLT/286 or Portable 386*

**Registers at call:**
AX = 4001h
CL = 00h timeout disabled else timeout in minutes

**Return Registers:**
AL = status
- 00h timeout modified
- 01h timeout cannot be modified
- 40h timeout cannot be modified

CL = timeout in minutes (00h if timeout disabled)

**Conflicts:** IBM Convertible (Chapter 4)
**See also:** Functions 4000h and 4601h

---

### INT 15h Function 4280h
### *ENTER STANDBY*

*Compaq SLT/286*

**Registers at call:**
AX = 4280h

**Return Registers:**
AH = 42h
CF clear if successful
CF set if unable to enter standby

**Conflicts:** IBM Convertible (Chapter 4)
**See also:** Function 4600h, APM Function 5307h (Chapter 19)

## INT 15h Function 4600h

*Compaq SLT/286*

### *READ POWER CONSERVATION/MODEM CONFIGURATION*

**Registers at call:**
AX = 4600h

**Return Registers:**
AH = modem configuration information
  bit 0 powerup state
    0 off
    1 on
  bit 1 modem installed
  bit 2 IRQ line assignment
    0 IRQ 4
    1 IRQ 3
  bit 3 COM port assignment
    0 = COM 2
    1 = COM 1
  bit 4 modem state
    0 not assigned
    1 assigned
  bit 5 modem is on
AL = power conservation status information
  bit 0 power source (0 internal, 1 external)
  bit 1-2 low battery state
    00 no low battery condition
    01 low battery 1
    10 reserved
    11 low battery 2
  bit 3-4 power conservation mode
    00 automatic
    01 on
    10 off
    11 reserved
BH = default system inactivity timeout (1-21 minutes)
BL = current system inactivity timeout (1-21 minutes)
CH = default video display inactivity timeout (1-63 minutes)
CL = current video display inactivity timeout (1-63 minutes)
DH = default fixed disk drive inactivity timeout (1-21 minutes)
DL = current fixed disk drive inactivity timeout (1-21 minutes)

**See also:** Functions 4280h and 4601h, INT 77h

---

## INT 15h Function 4601h

*Compaq SLT/286*

### *MODIFY POWER CONSERVATION/MODEM CONFIGURATION*

**Registers at call:**
AX = 4601h
BL = system inactivity timeout (1-21 minutes)
  FFh do not change
CL = video display inactivity timeout (1-63 minutes)
  FFh do not change
DL = current fixed disk drive inactivity

timeout (1-21 minutes)
FFh do not change
DH = modem state
00h turn modem OFF
01h turn modem ON
FFh do not change modem state

**Return Registers:**
CF clear if successful
AH = 00h
BL = current system inactivity timeout (1-21 minutes)
CL = current video display inactivity timeout (1-63 minutes)
DL = current fixed disk drive inactivity timeout (1-21 minutes)
DH = new modem state
00h modem turned OFF
01h modem turned ON
FFh modem state unchanged
CF set on error
AH = error
01h input is out of range
02h no modem present

**See also:** Function 4600h, INT 77h

---

### INT 15h Function E00Fh
### *MULTIPROCESSOR DISPATCH*

*Compaq Systempro*

**Registers at call:**
AX = E00Fh
ES:BX -> start of 2nd processor's execution

**Return Registers:**
AL = status
00h failure
0Fh successful

**See also:** Functions E10Eh and E200h

---

### INT 15h Function E10Eh
### *MULTIPROCESSOR END-OF-DISPATCH*

*Compaq Systempro*

**Registers at call:**
AX = E10Eh
ES:BX -> start of 2nd processor's execution

**Return Registers:**
AL = status
00h failure (not halted)
0Fh successful (halted)

**See also:** Functions E00Fh and E200h

---

### INT 15h Function E200h
### *MULTIPROCESSOR AVAILABLE*

*Compaq Systempro*

**Registers at call:**
AX = E200h

**Return Registers:**
AX = 8000h if 2nd processor available

**See also:** Functions E00Fh and E10Eh

---

### INT 16h Function F0h
### *SET CPU SPEED*

*Compaq 386*

**Registers at call:**
AH = F0h
AL = speed
00h equivalent to 6 MHz 80286 (COMMON)
01h equivalent to 8 MHz 80286 (FAST)
02h full 16 MHz (HIGH)
03h toggles between 8 MHz-equivalent and speed set by system board switch (AUTO or HIGH)
08h full 16 MHz except 8

MHz-equivalent during floppy disk access
09h specify speed directly
CX = speed value, 1 (slowest) to 50 (full), 3 = 8088

**Return Registers:**
none

**Conflicts:** TurboPower TSRs (Chapter 51)
**See also:** Functions F1h and F3h

---

### INT 16h Function F1h
*Compaq 386*

***READ CURRENT CPU SPEED***

**Registers at call:**
AH = F1h

**Return Registers:**
AL = speed code (see Function F0h)
if AL = 09h, CX = speed code

**See also:** Functions F0h and F3h

---

### INT 16h Function F2h
*Compaq 386*

***DETERMINE ATTACHED KEYBOARD TYPE***

**Registers at call:**
AH = F2h

**Return Registers:**
AL = type
00h if 11-bit AT keyboard is in use
01h if 9-bit PC keyboard is in use

---

### INT 16h Function F3h
*Compaq 80286s*

***SET CPU SPEED LIMIT (OVERRIDE JUMPER)***

**Registers at call:**
AH = F3h
AL = new speed limit
00h limit is 6 Mhz
01h limit is 8 Mhz/6 Mhz

**Return Registers:**
none

**Conflicts:** Norton Guides (Chapter 62)
**See also:** Functions F0h and F1h

---

### INT 16h Function F400h
*Compaq Systempro*

***CACHE CONTROLLER STATUS***

**Registers at call:**
AX = F400h

**Return Registers:**
AH = E2h
AL = status
00h not present
01h enabled
02h disabled

**See also:** Functions F401h and F402h

---

### INT 16h Function F401h
*Compaq Systempro*

***ENABLE CACHE CONTROLLER***

**Registers at call:**
AX = F401h

**Return Registers:**
AX = E201h

**See also:** Functions F400h and F402h

---

### INT 16h Function F402h
*Compaq Systempro*

***DISABLE CACHE CONTROLLER***

**Registers at call:**
AX = F402h

**Return Registers:**
AX = E202h

**See also:** Functions F400h and F401h

---

**INT 77h** *Compaq SLT/286 POWER CONSERVATION*

***IRQ15***

**Details:** May be masked by setting bit 7 on I/O port A1h.

**Conflicts:** Standard IRQ15 (Chapter 3)

**See also:** INT 15h Function 4600h

## Corona

---

**INT 10h Function 05h** *Corona/Cordata BIOS v4.10+*

***GRAPHICS BITMAP BUFFER***

**Purpose:** Determine or specify the address of the graphics bitmap buffer for video modes 60h and 61h.

| Registers at call: | Return Registers: |
|---|---|
| AH = 05h | |
| AL = function | |
| 00h set address of graphics bitmap buffer | |
| BX = segment of buffer | DX = segment of graphics bitmap buffer |
| 0Fh get address of graphics bitmap buffer | |

## Hewlett-Packard

Hewlett-Packard's Vectras are IBM compatible desktop machines, many of which support HP's Human Interface Link or HIL I/O peripheral interface.

The HP 95LX and its successor, the 100LX, are palmtop computers with an integrated infrared link for data transfer. The 95LX/100LX functions described here are those miscellaneous calls which don't belong to the system services described in Chapter 20.

---

**INT 06h** *HP 95LX*

***SLEEP/WAKEUP*** *Callout*

**Purpose:** This interrupt is called just before going into light or deep (shutdown) sleep and just after returning from light or deep sleep.

**Conflicts:** Hardware (Chapter 3)

**See also:** INT 0Bh, INT 15h Function 4DD4h, INT 15h Function 4Eh

---

**INT 0Bh** *HP 95LX*

***LOW-LEVEL KEYBOARD HANDLER***

**Purpose:** This is the lowest-level keyboard handler on an HP 95LX palmtop.

**Details:** Debounces key, places the keycode in I/O register 60h, and calls INT 09h.

**Conflicts:** IRQ3 (Chapter 3)

**See also:** INT 15h Function 4DD4h, Hardware INT 09h (Chapter 3), HP 95LX INT 0Dh (Chapter 3)

---

**INT 15h Function 4DD4h** *HP 95LX*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = 4DD4h | BX = 4850h ("HP") if HP 95LX<br>CX = *unknown* (0101h)<br>DL = *unknown* (00h) |

**See also:** INT 0Bh, HP 95LX INT 0Fh (Chapter 3), INT 5Fh Function 00h (Chapter 20),HP 95LX INT 60h Function 0100h (Chapter 20), HP 95LX INT 61h interface (Chapter 20)

### INT 15h Function 4Eh

*HP 95LX*

#### *ENABLE/DISABLE LIGHT SLEEP*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 4Eh | none |
| AL = light sleep | |
| 00h disabled | |
| 01h enabled | |

**Details:** When light sleep is disabled, the system will continue running at full speed; when enabled, it may automatically slow to conserve batteries.

**See also:** INT 06h, HP 95LX INT 60h Function 0100h (Chapter 20)

### INT 16h Function 6F00h

*HP HIL Vectras*

#### *HP HIL Extended BIOS INSTALLATION CHECK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 6F00h | BX = 4850h if present |
| BX = 0000h | |

**Details:** Called by recent MS Mouse drivers looking for an HP-HIL mouse. Supported by ES, QS, and RS series HP Vectras.

**See also:** Function 6F0Dh

### INT 16h Function 6F0Dh

*HP HIL Vectras*

#### *GET HIL Extended BIOS INTERRUPT NUMBER*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 6F0Dh | AH = interrupt number (default 6Fh, 02h means 6Fh as well) |

**Details:** Called by MS Windows HPSYSTEM.DRV and HPEBIOS.386 to support the HP-HIL input system.

**See also:** Function 6F0Eh,INT 6Fh Functions 22h and 24h

### INT 16h Function 6F0Eh

*HP HIL Vectras*

#### *SET HIL Extended BIOS INTERRUPT NUMBER*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 6F0Eh | AH = status (00h = successful) |
| BL = new interrupt number (60h-6Fh,78h-7Fh) | |

**Purpose:** Allows the HIL Extended BIOS software to use a non-default interrupt number in case of an interrupt conflict with another application.

**Details:** Called by MS Windows HPSYSTEM.DRV and HPEBIOS.386 to support the HP-HIL input system.

**See also:** Function 6F0Dh, INT 6Fh Functions 22h and 24h

### INT 6Fh Function 22h

*HP HIL Vectras*

#### *EXTENDED BIOS - READ CMOS MEMORY*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 22h | AH = status |
| BP = 0012h (ID of V_SYSTEM driver) | AL = byte read |
| BL = address of CMOS byte to read | BP, DS destroyed |

**Details:** Supported by ES, QS, and RS series HP Vectras.

**Conflicts:** 10-NET (NI)

**See also:** Function 24h

---

### INT 6Fh Function 24h
*HP HIL Vectras*

**EXTENDED BIOS - WRITE CMOS MEMORY**

**Registers at call:**
AH = 24h
BP = 0012h (ID of V_SYSTEM driver)
BL = address of CMOS byte to write
AL = new value

**Return Registers:**
AH = status
BP, DS destroyed

**See also:** Function 22h

## Phoenix

Phoenix Technologies, Ltd. is a major manufacturer of IBM-compatible BIOSes.

---

### INT 15h Function BCh
*Phoenix 386 BIOS*

**DETERMINE CPU SPEED**

**Registers at call:**
AH = BCh

**Return Registers:**
CF clear
BYTE 0040h:00B0h set to *unknown value* related to CPU speed

**Details:** Reads system timer channel 0 twice, then does calculations on returned values.

## Tandy

The following function appears to be specific to certain systems manufactured by Tandy, but it is not definitively known whether it is in fact a Tandy call. Additional Tandy-specific functions may be found in Chapter 8, Video.

---

### INT 15h Function E4h
*Tandy*

**UNDETERMINED FUNCTION**

**Registers at call:**
AH = E4h
AL = subfunction
  21h, 89h, 8Ah, 8Bh called by 386MAX v6.01
DL = *unknown value*

**Return Registers:**
DL = *00h if successful*

**Details:** The section of code in 386MAX which calls these functions also checks whether the ROM BIOS has both Tandy and Phoenix Technologies signatures if these calls fail.

## Victor

The Victor 9000 was another early IBM-compatible.

---

### INT DFh
*Victor 9000*

**SuperBIOS**

**Purpose:** This interrupt provides access to additional BIOS functions; the details of these functions are not known.

**Conflicts:** Screen Thief (Chapter 8), APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54)

## Zenith

The Zenith Z100 was an early MSDOS-compatible machine which distinguished itself through the use of the S100 bus and dual processors (using the second, Z80, processor for I/O processing).

**INT 40h** *Z100*

***Master 8259 - Parity error or S100 error***

**Conflicts:** Acorn (above), BIOS Diskette (Chapter 10)
**See also:** INT 41h, INT FFh

---

**INT 41h** *Z100*

***Master 8259 - Processor Swap***

**Conflicts:** Acorn (above), BIOS Data (Chapter 10), MS Windows (Chapter 57)
**See also:** INT 40h, INT 42h

---

**INT 42h** *Z100*

***Master 8259 - Timer***

**Conflicts:** Acorn (above), BIOS Video (Chapter 8), Western Digital SuperBIOS (Chapter 10)
**See also:** INT 41h, INT 43h

---

**INT 43h** *Z100*

***Master 8259 - Slave 8259 input***

**Details:** The slave interrupt controller runs in special fully nested mode.
**Conflicts:** Acorn (above), BIOS Video (Chapter 8)
**See also:** INT 42h, INT 44h

---

**INT 44h** *Z100*

***Master 8259 - Serial A***

**Conflicts:** Acorn (above), BIOS Video (Chapter 8), Virus (Chapter 59), NetWare (NI), IBM 3270-PC (NI)
**See also:** INT 43h, INT 45h

---

**INT 45h** *Z100*

***Master 8259 - Serial B***

**Conflicts:** Acorn (above)
**See also:** INT 44h, INT 46h

---

**INT 46h** *Z100*

***Master 8259 - Keyboard, Retrace, and Light Pen***

**Conflicts:** Acorn (above), BIOS Data (Chapter 10)
**See also:** INT 45h, INT 47h

---

**INT 47h** *Z100*

***Master 8259 - Printer***

**Conflicts:** Acorn (above), Western Digital SuperBIOS (Chapter 10), SQL Base (Chapter 54)
**See also:** INT 46h, INT 48h

---

**INT 48h** *Z100*

***Slave 8259 - S100 vectored line 0***

**Conflicts:** Acorn (above), Western Digital SuperBIOS (Chapter 10), PCjr (Chapter 13), Compaq UILIB (Chapter 54), Watstar PC Network (NI)
**See also:** INT 47h, INT 49h

---

**INT 49h** *Z100*

***Slave 8259 - S100 vectored line 1***

**Conflicts:** Acorn (above), Texas Instruments (Chapter 8), PCjr (Chapter 13), MAGic (Chapter 23), Watstar PC Network (NI)
**See also:** INT 48h, INT 4Ah

**INT 4Ah** *Z100*
***Slave 8259 - S100 vectored line 2***
**Conflicts:** Acorn (above), User Alarm (Chapter 4)
**See also:** INT 49h, INT 4Bh

**INT 4Bh** *Z100*
***Slave 8259 - S100 vectored line 3***
**Conflicts:** Acorn (above), CAM SCSI (Chapter 11), IBM SCSI (Chapter 11), VDS (Chapter 18)
**See also:** INT 4Ah, INT 4Ch

**INT 4Ch** *Z100*
***Slave 8259 - S100 vectored line 4***
**Conflicts:** Acorn (above)
**See also:** INT 4Bh, INT 4Dh

**INT 4Dh** *Z100*
***Slave 8259 - S100 vectored line 5***
**Conflicts:** none known
**See also:** INT 4Ch, INT 4Eh

**INT 4Eh** *Z100*
***Slave 8259 - S100 vectored line 6***
**Conflicts:** TI Professional PC (Chapter 10)
**See also:** INT 4Dh, INT 4Fh

**INT 4Fh** *Z100*
***Slave 8259 - S100 vectored line 7***
**Conflicts:** CAM SCSI (Chapter 11)
**See also:** INT 4Eh

**INT 69h** *Zenith AT BIOS*
***UNDETERMINED FUNCTION***
**Details:** This interrupt is called by the INT 09h handler.
**Conflicts:** ISR (Chapter 57), DECnet DOS (NI), 10-NET SYSSVC (NI)

**INT FFh** *Z100*
***WARM BOOT***
**Conflicts:** Protected-Mode Return (Chapter 4), QEMM (Chapter 39), Virus (Chapter 59)
**See also:** INT 40h

# CHAPTER 8

# *Video*

Virtually all IBM and video-compatible systems use the BIOS interface at INT 10h to control video actions. There are additional calls related to video services in other chapters. SuperVGA calls are covered in Chapter 9, TopView virtual screen support on INT 10h is covered in Chapter 44, and MultiDOS screen-related calls on INT 15h are listed in Chapter 45. Further, the Alloy multiuser networking systems (NTNX, 386/MultiWare) provide a fairly extensive set of video services; these are discussed in the companion book *Network Interrupts.*

Although not explicitly mentioned in the remainder of this chapter, all INT 10h calls potentially conflict with the coprocessor error interrupt described in Chapter 3.

## Standard BIOS Calls

### INT 10h Function 00h
### *SET VIDEO MODE*

**Purpose:** Specify the display mode for the currently active display adapter.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 00h | AL = video mode flag (Phoenix BIOS) |
| AL = new video mode (see Table 8-2) | 20h mode > 7 |
| | 30h modes 0-5 and 7 |
| | 3Fh mode 6 |
| | AL = CRT controller mode byte (Phoenix 386 BIOS v1.10) |

**Details:** IBM standard modes do not clear the screen if the high bit of AL is set (EGA or higher only).

The Tseng ET4000 chipset is used by the Orchid Prodesigner II, Diamond SpeedSTAR VGA, Groundhog Graphics Shadow VGA, Boca Super X VGA, Everex EV-673, and many others.

The installation check for Ahead adapters is the signature " AHEAD " at C000h:0025h.

The installation check for Paradise adapters is the signature " VGA= " at C000h:007Dh.

The installation check for the Oak Technologies OTI-037/057/067/077 chipsets is the signature " OAK VGA " at C000h:0008h.

The installation check for ATI adapters is the signature " 761295520 " at C000h:0031h; the byte at C000h:0043h indicates the chipset revision:

31h for 18800
32h for 18800-1
33h for 18800-2
34h for 18800-4
35h for 18800-5

the two bytes at C000h:0040h indicate the adapter type

" 22 " EGA Wonder

” 31 ” VGA Wonder
” 32 ” EGA Wonder800+

the byte at C000h:0042h contains feature flags
bit 1: mouse port present
bit 4: programmable video clock

the byte at C000h:0044h contains additional feature flags if the chipset byte is greater than 30h (see Table 8-1).

The installation check for Genoa video adapters is the signature 77h XXh 99h 66h at C000h:NNNNh, where NNNNh is stored at C000h:0037h and XXh is
00h for Genoa 6200/6300
11h for Genoa 6400/6600
22h for Genoa 6100
33h for Genoa 5100/5200
55h for Genoa 5300/5400

for SuperEGA BIOS v2.41+, C000h:0057h contains the product level

for SuperEGA BIOS v3.0+, C000h:0070h contains the signature “ EXTMODE ” , indicating support for extended modes.

**Conflicts:** Everex Function 0070h (below), Paradise Functions 007Eh and 007Fh

**See also:** Vendor-specific Functions 0070h and 007Eh (below), GRAFIX Function 40h (below), Functions 10E0h and 10F0h (Chapter 9), Video7 Function 6F05h (below), GO32 Function FFh (Chapter 37), Mouse INT 33h Function 0028h (Chapter 21), HP 95LX INT 5Fh Function 00h (Chapter 20), FastGraph/Light INT 62h Function 0001h (Chapter 52)

**8-1 Bitfields for ATI additional feature flags:**

| | |
|---|---|
| bit 0 | 70 Hz non-interlaced display |
| bit 1 | Korean (double-byte) characters |
| bit 2 | 45 MHz memory clock rather than 40 MHz |
| bit 3 | zero wait states |
| bit 4 | paged ROMs |
| bit 6 | no 8514/A monitor support |
| bit 7 | HiColor DAC |

**8-2 Values for video mode:**

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| 00h | = T | 40x25 | 8x8 | | 16gray | 8 | B800 | CGA,PCjr |
| | = T | 40x25 | 8x14 | | 16gray | 8 | B800 | EGA |
| | = T | 40x25 | 8x16 | | 16 | 8 | B800 | MCGA |
| | = T | 40x25 | 9x16 | | 16 | 8 | B800 | VGA |
| 01h | = T | 40x25 | 8x8 | | 16 | 8 | B800 | CGA,PCjr |
| | = T | 40x25 | 8x14 | | 16 | 8 | B800 | EGA |
| | = T | 40x25 | 8x16 | | 16 | 8 | B800 | MCGA |
| | = T | 40x25 | 9x16 | | 16 | 8 | B800 | VGA |
| 02h | = T | 80x25 | 8x8 | | 16gray | 4 | B800 | CGA,PCjr |
| | = T | 80x25 | 8x14 | | 16gray | 4 | B800 | EGA |
| | = T | 80x25 | 8x16 | | 16 | 4 | B800 | MCGA |
| | = T | 80x25 | 9x16 | | 16 | 4 | B800 | VGA |
| 03h | = T | 80x25 | 8x8 | | 16 | 4 | B800 | CGA,PCjr |
| | = T | 80x25 | 8x14 | | 16 | 4 | B800 | EGA |
| | = T | 80x25 | 8x16 | | 16 | 4 | B800 | MCGA |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| | = T | 80x25 | 9x16 | | 16 | 4 | B800 | VGA |
| 04h | = G | 40x25 | 8x8 | 320x200 | 4 | | B800 | CGA,PCjr, EGA,MCGA, VGA |
| 05h | = G | 40x25 | 8x8 | 320x200 | 4gray | | B800 | CGA,PCjr,EGA |
| | = G | 40x25 | 8x8 | 320x200 | 4 | | B800 | MCGA,VGA |
| 06h | = G | 80x25 | 8x8 | 640x200 | 2 | | B800 | CGA,PCjr, EGA,MCGA,VGA |
| 07h | = T | 80x25 | 9x14 | | mono | var | B000 | MDA,Hercules,EGA |
| | = T | 80x25 | 9x16 | | mono | | B000 | VGA |
| 08h | = T | 132x25 | 8x8 | | 16 | | B800 | ATI EGA/VGA Wonder [2] |
| | = T | 132x25 | 8x8 | | mono | | B000 | ATI EGA/VGA Wonder [2] |
| | = G | 20x25 | 8x8 | 160x200 | 16 | | | PCjr, Tandy 1000 |
| | = G | 90x43 | 8x8 | 720x352 | mono | | B000 | Hercules + MSHERC.COM |
| | = G | 90x45 | 8x8 | | mono | | B000 | Hercules + HERKULES [11] |
| 09h | = G | 40x25 | 8x8 | 320x200 | 16 | | | PCjr |
| 0Ah | = G | 80x25 | 8x8 | 640x200 | 4 | | | PCjr, Tandy 1000 |
| 0Bh | = | reserved (used internally by EGA BIOS) | | | | | | |
| | = G | | | 640x200 | 16 | | | Tandy 1000 SL/TL |
| 0Ch | = | reserved (used internally by EGA BIOS) | | | | | | |
| 0Dh | = G | 40x25 | 8x8 | 320x200 | 16 | 8 | A000 | EGA,VGA |
| 0Eh | = G | 80x25 | 8x8 | 640x200 | 16 | 4 | A000 | EGA,VGA |
| 0Fh | = G | 80x25 | 8x14 | 640x350 | mono | 2 | A000 | EGA,VGA |
| 10h | = G | 80x25 | 8x14 | 640x350 | 4 | 2 | A000 | 64k EGA |
| | = G | | | 640x350 | 16 | | A000 | 256k EGA,VGA |
| 11h | = G | 80x30 | 8x16 | 640x480 | mono | | A000 | VGA,MCGA, ATI EGA,ATI VIP |
| 12h | = G | 80x30 | 8x16 | 640x480 | 16/256k | | A000 | VGA,ATI VIP |
| | = G | 80x30 | 8x16 | 640x480 | 16/64 | | A000 | ATI EGA Wonder |
| | = G | | | 640x480 | 16 256K | | | UltraVision+ EGA |
| 13h | = G | 40x25 | 8x8 | 320x200 | 256/256k | | A000 | VGA,MCGA, ATI VIP |
| 14h | = T | 132x25 | Nx16 | | 16 | | | XGA, IBM Enhanced VGA [3] |
| | = T | 132x25 | 8x16 | 1056x400 | 16/256k | | | Cirrus CL-GD 5420/5422/5426 |
| | = G | 80x25 | 8x8 | 640x200 | | | | Lava Chrome II EGA |
| | = G | | | 640x400 | 16 | | | Tecmar VGA/AD |
| 15h | = G | 80x25 | 8x14 | 640x350 | | | | Lava Chrome II EGA |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| 16h | = G | 80x25 | 8x14 | 640x350 | | | | Lava Chrome II EGA |
| | = G | | | 800x600 | 16 | | | Tecmar VGA/AD |
| 17h | = T | 132x25 | | | | | | Tecmar VGA/AD |
| | = T | 80x43 | 8x8 | 640x350 | 16 | 4 | B800 | Tseng ET4000 BIOS [10] |
| | = G | 80x34 | 8x14 | 640x480 | | | | Lava Chrome II EGA |
| 18h | = T | 80x30 | 9x16 | 720x480 | 16 | 1 | A000 | Realtek RTVGA [12] |
| | = T | 132x25 | | | mono | | A000 | Cirrus 5320 chipset |
| | = T | 132x44 | 8x8 | | mono | | B000 | Tseng Labs EVA |
| | = T | 132x44 | 9x8 | 1188x352 | 4gray | 2 | B000 | Tseng ET3000 chipset |
| | = T | 132x44 | 8x8 | | 16/256 | 2 | B000 | Tseng ET4000 chipset |
| | = G | 80x34 | 8x14 | 640x480 | | | | Lava Chrome II EGA |
| | = G | | | 1024x768 | | 16 | | Tecmar VGA/AD |
| 19h | = T | 80x43 | 9x11 | 720x473 | 16 | 1 | A000 | Realtek RTVGA [12] |
| | = T | 132x25 | 8x14 | | mono | | B000 | Tseng Labs EVA |
| | = T | 132x25 | 9x14 | 1188x350 | 4gray | 4 | B000 | Tseng ET3000 chipset |
| | = T | 132x25 | 8x14 | | 16/256 | 4 | B000 | Tseng ET4000 chipset |
| | = T | 132x34 | | | mono | | B000 | Cirrus 5320 chipset |
| 1Ah | = T | 80x60 | 9x8 | 720x480 | 16 | 1 | A000 | Realtek RTVGA [12] |
| | = T | 132x28 | 8x13 | | mono | | B000 | Tseng Labs EVA |
| | = T | 132x28 | 9x13 | 1188x364 | 4gray | 4 | B000 | Tseng ET3000 chipset |
| | = T | 132x28 | 8x13 | | 16/256 | 4 | B000 | Tseng ET4000 chipset |
| | = T | 132x44 | | | mono | | B000 | Cirrus 5320 chipset |
| | = G | | | 640x350 | 256 | | | Tecmar VGA/AD |
| 1Bh | = T | 132x25 | 9x14 | 1188x350 | 16 | 1 | A000 | Realtek RTVGA [12] |
| | = G | | | 640x400 | 256 | | | Tecmar VGA/AD |
| 1Ch | = T | 132x25 | | | | | | Cirrus 5320 chipset |
| | = T | 132x30 | 9x16 | 1188x480 | 16 | 1 | A000 | Realtek RTVGA [12] |
| | = G | | | 640x480 | 256 | | | Tecmar VGA/AD |
| 1Dh | = T | 132x43 | | | | | | Cirrus 5320 chipset |
| | = T | 132x43 | 9x11 | 1188x473 | 16 | 1 | A000 | Realtek RTVGA [12] |
| | = G | | | 800x600 | 256 | | | Tecmar VGA/AD |
| 1Eh | = T | 132x44 | | | | | | Cirrus 5320 chipset |
| | = T | 132x60 | 9x8 | 1188x480 | 16 | 1 | A000 | Realtek RTVGA [12] |
| 1Fh | = G | 100x75 | 8x8 | 800x600 | 16 | 1 | A000 | Realtek RTVGA |
| 20h | = G | | | 240x128 | | | B000 | HP 95LX |
| | = G | 120x45 | 8x16 | 960x720 | 16 | 1 | A000 | Realtek RTVGA |
| 21h | = T | 132x60 | | | 16 | 2 | B800 | Tseng ET4000 chipset [10] |
| | = T | 132x44 | 9x9 | 1188x396 | 16/256k | | B800 | WD90C |
| | = T | 132x44 | 9x9 | | | 16 | B800 | Diamond Speedstar 24X |
| | = G | 80x43 | 8x8 | 720x348 | mono | | B000 | DESQview 2.x+Hercules [4] |
| | = G | 128x48 | 8x16 | 1024x768 | 16 | 1 | A000 | Realtek RTVGA [12] |
| 22h | = T | 132x43 | | | | | | Allstar Peacock (VGA) |
| | = T | 132x44 | 8x8 | | | | | Tseng Labs EVA |
| | = T | 132x44 | 9x8 | 1188x352 | 16/256k | 2 | B800 | Tseng ET3000 chipset |
| | = T | 132x44 | 8x8 | | 16/256k | 2 | B800 | Tseng ET4000 chipset |
| | = T | 132x44 | 8x8 | | | | | Ahead Systems EGA2001 |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| | = T | 132x44 | 8x8 | | 16 | 2 | B800 | Ahead B |
| | = T | 132x44 | | | | 16 | | Orchid Prodesigner VGA |
| | = G | 80x43 | 8x8 | 720x348 | mono | | B800 | DESQview 2.x+Hercules [4] |
| | = G | 96x64 | 8x16 | 768x1024 | 16 | 1 | A000 | Realtek RTVGA |
| 23h | = T | 132x25 | 6x14 | | | | | Tseng Labs EVA |
| | = T | 132x25 | 9x14 | 1188x350 | 16/256k | 4 | B800 | Tseng ET3000 chipset |
| | = T | 132x25 | 8x14 | | 16/256 | 4 | B800 | Tseng ET4000 chipset |
| | = T | 132x25 | 8x14 | | | | | Ahead Systems EGA2001 |
| | = T | 132x25 | 8x14 | | 16 | 4 | B800 | Ahead B |
| | = T | 132x25 | 8x8 | | 16 | | B800 | ATI EGA Wonder, ATI VIP |
| | = T | 132x25 | | | | | | Cirrus 5320 chipset |
| | = T | 132x28 | | | | | | Allstar Peacock (VGA) |
| | = T | 132x28 | | | 16 | | | Orchid Prodesigner VGA |
| | = G | 128x48 | 8x16 | 1024x768 | 4 | 1 | A000 | Realtek RTVGA |
| 24h | = T | 132x25 | | | | | | Allstar Peacock (VGA) |
| | = T | 132x25 | | | 16 | | | Orchid Prodesigner VGA |
| | = T | 132x28 | 6x13 | | | | | Tseng Labs EVA |
| | = T | 132x28 | 9x13 | 1188x364 | 16/256k | 4 | B800 | Tseng ET3000 chipset |
| | = T | 132x28 | 8x13 | | 16/256k | 4 | B800 | Tseng ET4000 chipset |
| | = T | 132x28 | 8x12 | | 16 | 1 | B800 | Ahead B |
| | = T | 132x28 | | | | | | Cirrus 5320 chipset |
| | = G | 64x32 | 8x16 | 512x512 | 256 | 1 | A000 | Realtek RTVGA |
| 25h | = G | 80x60 | 8x8 | 640x480 | | | | Tseng Labs EVA |
| | = G | 80x60 | 8x8 | 640x480 | 16/256k | 1 | A000 | Tseng ET3000/ 4000 chipset |
| | = G | 640x480 | 16 | | | | | VEGA VGA |
| | = G | 80x60 | 8x8 | 640x480 | 16 | | A000 | Orchid Prodesigner VGA |
| | = G | 80x60 | 8x8 | 640x480 | 16 | | A000 | Ahead B (same as 26h) |
| | = G | 640x480 | 16 | | | | | NEC GB-1 |
| | = G | 640x480 | 16 | | | | | Cirrus 5320 chipset |
| | = G | 640x400 | 256 | | | | | Realtek RTVGA |
| 26h | = T | 80x60 | 8x8 | | | | | Tseng Labs EVA |
| | = T | 80x60 | 8x8 | 640x480 | 16/256k | 3 | B800 | Tseng ET3000/ 4000 chipset |
| | = T | 80x60 | | | | | | Allstar Peacock (VGA) |
| | = T | 80x60 | | | 16 | | | Orchid ProDesigner VGA |
| | = G | 80x60 | 8x8 | 640x480 | | | | Ahead Systems EGA2001 |
| | = G | 80x60 | 8x8 | 640x480 | 16 | 1 | A000 | Ahead B (same as 25h) |
| | = G | 640x480 | 256 | | | | | Realtek RTVGA |
| 27h | = T | 132x25 | 8x8 | | mono | | B000 | ATI EGA Wonder, ATI VIP |
| | = G | 720x512 | 16 | | | | | VEGA VGA |
| | = G | 720x512 | 16 | | | | | Genoa |
| | = G | 100x75 | 8x8 | 800x600 | 256 | 1 | A000 | Realtek RTVGA [12] |
| 28h | = T | *unknown* | | | | | | VEGA VGA |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| | = G | 1024x768 | 256 | | | | | Realtek RTVGA (1meg) |
| 29h | = G | 800x600 | 16 | | | | | VEGA VGA |
| | = G | 100x37 | 8x16 | 800x600 | 16 | | A000 | Orchid |
| | = G | 800x600 | 16 | | | | A000 | STB,Genoa,Sigma |
| | = G | 800x600 | 16 | | | | | Allstar Peacock (VGA) |
| | = G | 100x37 | 8x16 | 800x600 | 16/256k | 1 | A000 | Tseng ET3000/ 4000 chipset |
| | = G | 800x600 | | | | | | EIZO MDB10 |
| | = G | 800x600 | | | 16 | | | Cirrus 5320 chipset |
| | = G | 1024x1024 | | 256 | | | Realtek | RTVGA BIOS v3.C10 |
| 2Ah | = T | 100x40 | | | | | | Allstar Peacock (VGA) |
| | = T | 100x40 | 8x16 | | 16 | | | Orchid Prodesigner VGA |
| | = T | 100x40 | 8x15 | 800x600 | 16/256k | 4 | B800 | Tseng ET3000/ 4000 chipset |
| | = G | 1280x1024 | | | 16 | | | Realtek RTVGA |
| 2Dh | = G | 640x350 | | | 256 | | | VEGA VGA |
| | = G | 40x350 | | | 256/256k | | A000 | Orchid, Genoa, STB |
| | = G | 80x25 | 8x14 | 640x350 | 256/256k | 1 | A000 | Tseng ET3000/ 4000 chipset |
| | = G | 640x350 | | | 256 | | | Cirrus 5320 chipset |
| 2Eh | = G | 640x480 | | | 256 | | | VEGA VGA |
| | = G | 80x30 | 8x16 | 640x480 | 256/256k | | A000 | Orchid |
| | = G | 640x480 | 256/256k | | | | A000 | STB,Genoa,Sigma |
| | = G | 80x30 | 8x16 | 640x480 | 256/256k | 1 | A000 | Tseng ET3000/ 4000 chipset |
| 2Fh | = T | 160x50 | 8x8 | 1280x400 | 16 | 4 | B800 | Ahead B (Wizard/3270) |
| | = G | 720x512 | | | 256 | | | VEGA VGA |
| | = G | 720x512 | | | 256 | | | Genoa |
| | = G | 80x25 | 8x16 | 640x400 | 256/256k | 1 | A000 | Tseng ET4000 chipset |
| 30h | = G | 800x600 | | | 256 | | | VEGA VGA |
| | = G | 100x37 | 8x16 | 800x600 | 256/256k | | A000 | Orchid |
| | = G | 800x600 | 256/256k | | | | A000 | STB,Genoa,Sigma |
| | = G | 720x350 | 2 | | | | | 3270 PC |
| | = G | 800x600 | 256 | | | | | Cardinal |
| | = G | *unknown* | | | | | B800 | AT&T 6300 |
| | = G | 100x37 | 8x16 | 800x600 | 256/256k | 1 | A000 | Tseng ET3000/ 4000 chipset |
| 32h | = T | 80x34 | 8x10 | | 16 | 4 | B800 | Ahead B (Wizard/3270) |
| 33h | = T | 132x44 | 8x8 | | 16 | | | B800 ATI EGA Wonder, ATI VIP |
| | = T | 80x34 | 8x8 | | 16 | 4 | B800 | Ahead B (Wizard/3270) |
| 34h | = T | 80x66 | 8x8 | | 16 | 4 | B800 | Ahead B (Wizard/3270) |
| 36h | = G | 960x720 | 16 | | | | | VEGA VGA |
| | = G | 960x720 | 16 | | | | | STB |
| | = G | 960x720 | 16 | | | | | Tseng ET3000 only |
| 37h | = T | 132x44 | 8x8 | | mono | | B800 | ATI EGA Wonder, ATI VIP |

| | text/ grph | text resol | pixel box | pixel resoltn | colors | disply pages | scrn addr | system |
|---|---|---|---|---|---|---|---|---|
| | = G | 1024x768 | 16 | | | | | VEGA VGA |
| | = G | 128x48 | 8x16 | 1024x768 | 16 | | A000 | Orchid |
| | = G | 1024x76 | 8 | 16 | | | A000 | STB,Genoa,Sigma |
| | = G | 1024x76 | 8 | 16 | | | | Definicon |
| | = G | 128x48 | 8x16 | 1024x768 | 16 | 1 | A000 | Tseng ET3000/ 4000 chipset |
| 38h | = G | 1024x768 | 256 | | | | | STB VGA/EM-16 Plus (1MB) |
| | = G | 128x48 | 8x16 | 1024x768 | 256/256k | 1 | A000 | Tseng ET4000 chipset |
| | = G | 1024x768 | 256 | | | | | Orchid ProDesigner II |
| 3Dh | = G | 1280x1024 | 16 | | | | | Definicon |
| | = G | 128x64 | 8x16 | 1280x1024 | 16 | 1 | A000 | Tseng ET4000 v3.00 [1,7] |
| 3Eh | = G | 1280x961 | | | 16 | | | Definicon |
| 40h | = T | 80x43 | | | | | | VEGA VGA, Tecmar VGA/AD |
| | = T | 80x43 | | | | | | Video7 V-RAM VGA |
| | = T | 80x43 | | | | | | Tatung VGA |
| | = T | 100x30 | | | 16 | | | MORSE VGA |
| | = T | 100x30 | | | | | | Cirrus 510/520 chipset |
| | = T | 80x25 | | 720x350 | mono | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | 80x25 | 8x16 | 640x400 | 2 | 1 | B800 | AT&T 6300, AT&T VDC600 |
| | = G | 80x25 | 8x16 | 640x400 | 2 | 1 | B800 | Compaq Portable |
| 41h | = T | 132x25 | | | | | | VEGA VGA |
| | = T | 132x25 | | | | | | Tatung VGA |
| | = T | 132x25 | | | | | | Video7 V-RAM VGA |
| | = T | 100x50 | | | 16 | | | MORSE VGA |
| | = T | 100x50 | | | | | | Cirrus 510/520 chipset |
| | = T | 80x34 | 9x14 | 720x476 | 16/256k | | B800 | WD90C |
| | = T | 80x34 | 9x14 | | 16 | | B800 | Diamond Speedstar 24X |
| | = G | | | 640x200 | 16 | 1 | | AT&T 6300 |
| | = G | 80x25 | | 720x348 | mono | | B000 | Genoa SuperEGA BIOS 3.0+ |
| 42h | = T | 132x43 | | | | | | VEGA VGA |
| | = T | 132x43 | | | | | | Tatung VGA |
| | = T | 132x43 | | | | | | Video7 V-RAM VGA |
| | = T | 80x34 | 9x10 | | 4 | 4 | B800 | Ahead B (Wizard/3270) |
| | = T | 100x60 | | | 16 | | | MORSE VGA |
| | = T | 100x60 | | | | | | Cirrus 510/520 chipset |
| | = G | 80x25 | 8x16 | 640x400 | 16 | | | AT&T 6300, AT&T VDC600 |
| | = G | 80x25 | | 720x348 | mono | | B800 | Genoa SuperEGA BIOS 3.0+ |
| 43h | = T | 80x60 | | | | | | VEGA VGA |
| | = T | 80x60 | | | | | | Tatung VGA |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| | = T | 80x60 | | | | | | Video7 V-RAM VGA |
| | = T | 80x45 | 9x8 | | 4 | 4 | B800 | Ahead B (Wizard/3270) |
| | = T | 100x75 | | | 16 | | | MORSE VGA |
| | = T | 80x29 | | 720x348 | mono | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | 640x200 of 640x400 | | | | | | viewport AT&T 6300 (unsupported) |
| 44h | = | disable VDC and DEB output | | | | | | AT&T 6300 |
| | = T | 100x60 | | | | | | VEGA VGA |
| | = T | 100x60 | | | | | | Tatung VGA |
| | = T | 100x60 | | | | | | Video7 V-RAM VGA |
| | = T | 80x32 | | 720x352 | mono | | | Genoa SuperEGA BIOS 3.0+ |
| 45h | = T | 132x28 | | | | | | Tatung VGA |
| | = T | 132x28 | | | | | | Video7 V-RAM VGA |
| | = T | 80x44 | | 720x352 | mono | | | Genoa SuperEGA BIOS 3.0+ |
| 46h | = T | 132x25 | 8x14 | | mono | | | Genoa 6400 |
| | = T | 132x25 | 9x14 | | mono | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | 100x40 | 8x15 | 800x600 | 2 | | | AT&T VDC600 |
| 47h | = T | 132x29 | 8x12 | | mono | | | Genoa 6400 |
| | = T | 132x29 | 9x12 | | mono | | | Genoa SuperEGA BIOS 3.0+ |
| | = T | 132x28 | 9x16 | 1188x448 | 16/256k | | B800 | WD90C |
| | = T | 132x28 | 9x16 | | 16 | | B800 | Diamond Speedstar 24X |
| | = G | 100x37 | 8x16 | 800x600 | 16 | | | AT&T VDC600 |
| 48h | = T | 132x32 | 8x12 | | mono | | | Genoa 6400 |
| | = T | 132x32 | 9x11 | | mono | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | 80x50 | 8x8 | 640x400 | 2 | | B800 | AT&T 6300, AT&T VDC600 |
| 49h | = T | 132x44 | 8x8 | | mono | | | Genoa 6400 |
| | = T | 132x44 | 9x8 | | mono | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | 80x30 | 8x16 | 640x480 | | | | Lava Chrome II EGA |
| 4Dh | = T | 120x25 | | | | | | VEGA VGA |
| 4Eh | = T | 120x43 | | | | | | VEGA VGA |
| | = T | 80x60 | 8x8 | | 16/256k | | B800 | Oak OTI-067/ OTI-077 [8] |
| 4Fh | = T | 132x25 | | | | | | VEGA VGA |
| | = T | 132x60 | | | | | | some Oak Tech VGA [8] |
| 50h | = T | 80x30 | 8x16 | | 16/256k | | B800 | Trident TVGA 8800/8900 |
| | = T | 80x34 | | | | | | Lava Chrome II EGA |
| | = T | 80x43 | | | mono | | | VEGA VGA |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| | = T | 132x25 | 9x14 | | mono | | | Ahead Systems EGA2001 |
| | = T | 132x25 | 9x14 | | 4 | 4 | B800 | Ahead B |
| | = T | 132x25 | 8x14 | | 16 | 8 | B800 | OAK Technologies VGA-16 |
| | = T | 132x25 | 8x14 | | 16/256k | | B800 | Oak OTI-037/067/077 [8] |
| | = T | 132x25 | 8x14 | 1056x350 | 16 | 8 | B800 | UM587 chipset |
| | = T | 132x30 | | | 16 | | | MORSE VGA |
| | = T | 132x30 | | | | | | Cirrus 510/520 chipset |
| | = G | 80x30 | 8x16 | 640x480 | 16 | | | Paradise EGA-480 |
| | = G | 80x30 | 8x16 | 640x480 | 16 | | | NEL Electronics BIOS |
| | = G | 640x480 | *mono* | | | | | Taxan 565 EGA |
| | = G | 40x25 | 8x8 | 320x200 | | | | Genoa SuperEGA BIOS 3.0+ |
| 51h | = T | 80x30 | 8x16 | | | | | Paradise EGA-480 |
| | = T | 80x30 | 9x16 | | | | | NEL Electronics BIOS |
| | = T | 80x30 | | | | | | Lava Chrome II EGA |
| | = T | 80x43 | 8x11 | | 16/256k | | B800 | Trident TVGA 8800/8900 |
| | = T | 132x25 | | | mono | | | VEGA VGA |
| | = T | 132x28 | 9x12 | | 4 | 4 | B800 | Ahead B |
| | = T | 132x43 | 8x8 | | 16 | 5 | B800 | OAK Technologies VGA-16 |
| | = T | 132x43 | 8x8 | | 16/256k | | B800 | Oak OTI-037/067/077 |
| | = T | 132x43 | 8x8 | 1056x344 | 16 | 5 | B800 | UM587 chipset |
| | = T | 132x50 | | | 16 | | | MORSE VGA |
| | = T | 132x50 | | | | | | Cirrus 510/520 chipset |
| | = G | 80x34 | 8x14 | 640x480 | 16 | | | ATI EGA Wonder |
| | = G | 80x25 | 8x8 | 640x200 | | | | Genoa SuperEGA BIOS 3.0+ |
| 52h | = T | 80x60 | | | | | | Lava Chrome II EGA |
| | = T | 80x60 | 8x8 | | 16/256k | | B800 | Trident TVGA 8800/8900 |
| | = T | 132x43 | | | mono | | | VEGA VGA |
| | = T | 132x44 | 9x8 | | mono | | | Ahead Systems EGA2001 |
| | = T | 132x44 | 9x8 | | 4 | 2 | B800 | Ahead B |
| | = T | 132x60 | | | 16 | | | MORSE VGA |
| | = T | 132x60 | | | | | | Cirrus 510/520 chipset |
| | = G | 94x29 | 8x14 | 752x410 | 16 | | | ATI EGA Wonder |
| | = G | 100x75 | 8x8 | 800x600 | 16 | 1 | A000 | OAK Technologies VGA-16 |
| | = G | 100x75 | 8x8 | 800x600 | 16 | | A000 | Oak OTI-037 chipset [8] |
| | = G | 100x37 | 8x16 | 800x600 | 16 | | A000 | Oak OTI-067/077 chips [8] |
| | = G | 100x75 | 8x8 | 800x600 | 16 | | A000 | UM587 chipset |
| | = G | 128x30 | 8x16 | 1024x480 | 16 | | | NEL Electronics BIOS |
| 53h | = T | 80x25 | 8x16 | | | | | NEL Electronics BIOS |
| | = T | 80x60 | | | 16 | | | MORSE VGA |
| | = T | 80x60 | | | | | | Cirrus 510/520 chipset |
| | = T | 132x25 | 8x14 | | 16/256k | | B800 | Trident TVGA 8800/8900 |
| | = T | 132x43 | | | | | | Lava Chrome II EGA |
| | = G | | | 640x480 | 256 | | | Oak VGA |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| | = G | 80x30 | 8x16 | 640x480 | 256 | | A000 | Oak OTI-067/ OTI-077 [8] |
| | = G | 100x40 | 8x14 | 800x560 | 16 | | | ATI EGA Wonder, ATI VIP |
| | = G | | | | | | | AX PC |
| 54h | = T | 132x25 | | | | | | Lava Chrome II EGA |
| | = T | 132x30 | 8x16 | | 16/256k | | B800 | Trident TVGA 8800/ 8900 |
| | = T | 132x43 | 8x8 | | | | | Paradise EGA-480 |
| | = T | 132x43 | 8x8 | | | | | NEL Electronics BIOS |
| | = T | 132x43 | 7x9 | | 16/256k | | B800 | Paradise VGA |
| | = T | 132x43 | 8x9 | | 16/256k | | B800 | Paradise VGA on multisync |
| | = T | 132x43 | | | | | | Taxan 565 EGA |
| | = T | 132x43 | | | | | | AST VGA Plus |
| | = T | 132x43 | | | | | | Hewlett-Packard D1180A |
| | = T | 132x43 | 7x9 | | 16 | | | AT&T VDC600 |
| | = T | 132x43 | 9x9 | 1188x387 | 16/256k | | B800 | WD90C |
| | = T | 132x43 | 9x9 | | 16 | | B800 | Diamond Speedstar 24X |
| | = T | 132x43 | 8x8 | 1056x350 | 16/256k | | | Cirrus CL-GD5420/ 5422/5426 |
| | = T | 132x50 | 8x8 | | 16 | | A000 | NCR 77C22 [9] |
| | = G | 100x42 | 8x14 | 800x600 | 16 | | A000 | ATI EGA Wonder, VGA Wonder |
| | = G | 100x42 | 8x14 | 800x600 | 16 | | A000 | ATI Ultra 8514A, ATI XL |
| | = G | | | 800x600 | 256 | | A000 | Oak VGA |
| | = G | 100x37 | 8x16 | 800x600 | 256 | | A000 | Oak OTI-067/077 chips [8] |
| 55h | = T | 80x66 | 8x8 | | 16/256k | | A000 | ATI VIP |
| | = T | 132x25 | 8x14 | | | | | Paradise EGA-480 |
| | = T | 132x25 | 8x14 | | | | | NEL Electronics BIOS |
| | = T | 132x25 | 7x16 | | 16/256k | | B800 | Paradise VGA |
| | = T | 132x25 | 8x16 | | 16/256k | | B800 | Paradise VGA on multisync |
| | = T | 132x25 | | | | | | Taxan 565 EGA |
| | = T | 132x25 | | | | | | AST VGA Plus |
| | = T | 132x25 | | | | | | Hewlett-Packard D1180A |
| | = T | 132x25 | 7x16 | | 16 | | | AT&T VDC600 |
| | = T | 132x25 | 8x16 | | 16 | | A000 | NCR 77C22 [9] |
| | = T | 132x25 | 9x16 | 1188x400 | 16/256k | | B800 | WD90C |
| | = T | 132x25 | 9x16 | | 16 | | B800 | Diamond Speedstar 24X |
| | = T | 132x25 | 8x14 | 1056x350 | 16/256k | | | Cirrus CL-GD5420/ 5422/5426 |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| | = T | 132x43 | 8x11 | | 16/256k | | B800 | Trident TVGA 8800/8900 |
| | = G | 94x29 | 8x14 | 752x410 | | | | Lava Chrome II EGA |
| | = G | 128x48 | 8x16 | 1024x768 | 16/256k | | A000 | ATI VGA Wonder v4+[5] |
| | = G | 1024x768 | | | 16/256k | | | ATI VGA Wonder Plus |
| | = G | 1024x768 | | | 16/256k | | | ATI Ultra 8514A,ATI XL |
| | = G | 128x48 | 8x16 | 1024x768 | 4 | | A000 | Oak OTI-067/077 chips [8] |
| 56h | = T | 132x43 | 8x8 | | *3* 2 | | B000 | NSI Smart EGA+ |
| | = T | 132x43 | 7x9 | | 4 | | B000 | Paradise VGA |
| | = T | 132x43 | 8x9 | | 4 | | B000 | Paradise VGA on multisync |
| | = T | 132x43 | | | mono | | | Taxan 565 EGA |
| | = T | 132x43 | 7x9 | | 2 | | | AT&T VDC600 |
| | = T | 132x43 | 9x8 | | | | | NEL Electronics BIOS |
| | = T | 132x50 | 8x8 | | 4 | | A000 | NCR 77C22 [9] |
| | = T | 132x60 | 8x8 | | 16/256k | | B800 | Trident TVGA 8800/8900 |
| | = G | 1024x768 | 16 | | | | A000 | Oak VGA |
| | = G | 128x48 | 8x16 | 1024x768 | 16 | | A000 | Oak OTI-067/077 chips [8] |
| 57h | = T | 132x25 | 8x14 | | *3* 4 | | B000 | NSI Smart EGA+ |
| | = T | 132x25 | 7x16 | | 4 | | B000 | Paradise VGA |
| | = T | 132x25 | 8x16 | | 4 | | B000 | Paradise VGA on multisync |
| | = T | 132x25 | 9x14 | | | | | NEL Electronics BIOS |
| | = T | 132x25 | | | mono | | | Taxan 565 EGA |
| | = T | 132x25 | 7x16 | | 2 | | | AT&T VDC600 |
| | = T | 132x25 | 9x14 | | 16/256k | | B800 | Trident TVGA 8800/ 8900 |
| | = T | 132x25 | 8x16 | | 4 | | A000 | NCR 77C22 [9] |
| | = G | 96x48 | 8x16 | 768x1024 | 16 | | A000 | Oak OTI-067/077 chips [8] |
| 58h | = T | 80x33 | 8x14 | | 16 | | B800 | ATI EGA Wonder, ATI VIP |
| | = T | 80x32 | 9x16 | | 16 | | | Genoa 6400 |
| | = T | 80x43 | 8x8 | | | | | NEL Electronics BIOS |
| | = T | 132x30 | 9x16 | | 16/256k | | B800 | Trident TVGA 8800/ 8900 |
| | = G | 100x75 | 8x8 | 800x600 | 16/256k | | A000 | Paradise VGA |
| | = G | 100x75 | 8x8 | 800x600 | 16 | | | AT&T VDC600 |
| | = G | 100x75 | 8x8 | 800x600 | 16 | | A000 | NCR 77C22 [9] |
| | = G | 100x75 | 8x8 | 800x600 | 16 | | | Diamond Speedstar 24X |
| | = G | 100x75 | 8x8 | 800x600 | 16/256k | | A000 | Paradise VGA, WD90C |
| | = G | | | 800x600 | 16 | | | AST VGA Plus, Compaq VGA |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| | = G | | | 800x600 | 16 | | | Dell VGA |
| | = G | | | 800x600 | 16 | | | Hewlett-Packard D1180A |
| | = G | | | 800x600 | | | | ELT VGA PLUS 16 |
| | = G | 100x37 | 8x16 | 800x600 | 16/256k | | A000 | Cirrus CL-GD5420/ 5422/5426 |
| | = G | 160x64 | 8x16 | 1280x1024 | 16 | | A000 | Oak OTI-077 chipset [8] |
| 59h | = T | 80x43 | 9x8 | | | | | NEL Electronics BIOS |
| | = T | 80x66 | 8x8 | | 16/256k | | A000 | ATI VIP |
| | = T | 132x43 | 9x11 | | 16/256k | | B800 | Trident TVGA 8800/8900 |
| | = G | 100x75 | 8x8 | 800x600 | 2 | | | A000 Paradise VGA |
| | = G | 100x75 | 8x8 | 800x600 | 2 | | | AT&T VDC600 |
| | = G | | | 800x600 | 2 | | | AST VGA Plus, Compaq VGA |
| | = G | | | 800x600 | 2 | | | Dell VGA |
| | = G | | | 800x600 | 2 | | | Hewlett-Packard D1180A |
| | = G | 100x75 | 8x8 | 800x600 | 2 | | A000 | NCR 77C22 [9] |
| | = G | 128x48 | 8x16 | 1024x768 | 256 | | A000 | Oak OTI-077 chipset [8] |
| 5Ah | = T | 80x60 | 8x8 | | | | | NEL Electronics BIOS |
| | = T | 132x60 | 9x8 | | 16/256k | | B800 | Trident TVGA 8800/8900 |
| | = G | 128x48 | 8x16 | 1024x768 | | 2 | A000 | NCR 77C22 [9] |
| 5Bh | = T | 80x30 | 8x16 | | | | B800 | ATI VGA Wonder (undoc) |
| | = G | | | 640x350 | 256 | | | Genoa 6400 |
| | = G | 80x25 | 8x16 | 640x400 | 32K | | A000 | Oak OTI-067/077 chips [8] |
| | = G | | | 800x600 | 16 | | | Maxxon, SEFCO TVGA, Imtec |
| | = G | 100x75 | 8x8 | 800x600 | 16/256k | | A000 | Trident TVGA 8800, 8900 |
| | = G | | | 800x600 | | | | Vobis MVGA |
| | = G | 100x37 | 8x16 | 800x600 | | | | NEL Electronics BIOS |
| | = G | 128x48 | 8x16 | 1024x768 | 16 | | A000 | NCR 77C22 [1,9] |
| 5Ch | = T | 100x37 | 8x16 | | | | | NEL Electronics BIOS |
| | = G | | | 640x400 | 256 | | | Logix, ATI Prism Elite |
| | = G | | | 640x400 | 256 | | | Maxxon, SEFCO TVGA, Imtec |
| | = G | 80x25 | 8x16 | 640x400 | 256/256k | | A000 | Zymos Poach, Hi Res 512 |
| | = G | 80x25 | 8x16 | 640x400 | 256/256k | | A000 | Trident TVGA 8800/8900 |
| | = G | 80x30 | 8x16 | 640x480 | 256 | | | Genoa 6400 |
| | = G | 80x30 | 8x16 | 640x480 | 32K | | A000 | Oak OTI-077 chipset [8] |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| | = G | 100x75 | 8x8 | 800x600 | 256 | | A000 | NCR 77C22 [9] |
| | = G | 100x75 | 8x8 | 800x600 | 256/256k | | A000 | WD90C |
| | = G | 100x75 | 8x8 | 800x600 | 256/256k | | A000 | Diamond Speedstar 24X |
| | = G | 100x37 | 8x16 | 800x600 | 256/256k | | A000 | Cirrus CL-GD5420/5422/5426 |
| 5Dh | = T | 100x75 | 8x8 | | | | | NEL Electronics BIOS |
| | = G | | | 640x480 | 256 | | | Logix, ATI Prism Elite |
| | = G | | | 640x480 | 256 | | | Maxxon, SEFCO TVGA, Imtec |
| | = G | 80x30 | 8x16 | 640x480 | 256/256k | | A000 | Zymos Poach, Hi Res 512 |
| | = G | 80x30 | 8x16 | 640x480 | 256/256k | | A000 | Trident TVGA 8800 (512K) |
| | = G | 128x48 | 8x16 | 1024x768 | 16 | | A000 | NCR 77C22 [9] |
| | = G | 128x48 | 8x16 | 1024x768 | 16/256k | | A000 | WD90C |
| | = G | 128x48 | 8x16 | 1024x768 | 16 | | | Diamond Speedstar 24X |
| | = G | 128x48 | 8x16 | 1024x768 | 16/256k | | A000 | Cirrus CL-GD5420/5422/5426 |
| 5Eh | = G | | | 640x400 | 256 | | | Paradise VGA, VEGA VGA |
| | = G | | | 640x400 | 256 | | | AST VGA Plus, NCR 77C22 |
| | = G | | | 640x400 | 256 | | | Compaq VGA, Dell VGA |
| | = G | 80x25 | 8x16 | 640x400 | 256 | | | AT&T VDC600 |
| | = G | 80x25 | 8x16 | 640x400 | 256 | | A000 | NCR 77C22 [9] |
| | = G | 80x25 | 8x16 | 640x400 | 256/256k | | A000 | WD90C |
| | = G | 80x25 | 8x16 | 640x400 | 256/256k | | A000 | Diamond Speedstar 24X |
| | = G | | | 800x600 | 16 | | | Logix, ATI Prism Elite |
| | = G | 100x37 | 8x16 | 800x600 | 16 | | | NEL Electronics BIOS |
| | = G | 100x75 | 8x8 | 800x600 | 256 | | | Genoa 6400 |
| | = G | 100x75 | 8x8 | 800x600 | 256/256k | | A000 | Zymos Poach, Trident 8900 |
| | = G | 100x75 | 8x8 | 800x600 | 256/256k | | A000 | Hi Res 512 |
| 5Fh | = G | | | 640x480 | 256 | | | Paradise VGA |
| | = G | | | 640x480 | 256 | | | AST VGA Plus, NCR 77C22 |
| | = G | | | 640x480 | 256 | | | Compaq VGA, Dell VGA |
| | = G | | | 640x480 | 256 | | | Hewlett-Packard D1180A |
| | = G | 80x30 | 8x16 | 640x480 | 256 | | | AT&T VDC600 (512K) |
| | = G | 80x30 | 8x16 | 640x480 | 256 | | A000 | NCR 77C22 [9] |
| | = G | 80x30 | 8x16 | 640x480 | 256/256k | | A000 | WD90C |
| | = G | 80x30 | 8x16 | 640x480 | 256/256k | | A000 | Diamond Speedstar 24X |
| | = G | 80x30 | 8x16 | 640x480 | 256/256k | | A000 | Cirrus CL-GD5420/5422/5426 |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| | = G | | | 1024x768 | 16 | | | Logix, ATI Prism Elite |
| | = G | | | 1024x768 | 16 | | | Maxxon, Imtec |
| | = G | 128x48 | 8x16 | 1024x768 | 16 | | | Genoa 6400 |
| | = G | 128x48 | 8x16 | 1024x768 | 16/256k | | A000 | Zymos Poach, Hi Res 512 |
| | = G | 128x48 | 8x16 | 1024x768 | 16/256k | | A000 | Trident TVGA 88/ 8900 512K |
| 60h | = T | 132x25 | 8x14 | | 16/64 | 8 | B800 | Quadram Ultra VGA |
| | = T | 132x25 | 8x14 | | 16 | | | Genoa 6400 |
| | = T | 132x25 | 8x14 | | 16 | | | Genoa SuperEGA BIOS 3.0+ |
| | = T | 132x25 | | | | | | Cirrus 5320 chipset |
| | = G | 80x? | | ?x400 | | | | Corona/Cordata BIOS 4.10+ |
| | = G | 80x25 | 8x16 | 640x400 | 256 | 1 | A000 | Ahead A, Ahead B |
| | = G | | | 752x410 | | | | VEGA VGA |
| | = G | | | 752x410 | 16 | | | Tatung VGA |
| | = G | | | 752x410 | 16 | | | Video7 V-RAM VGA |
| | = G | 128x48 | 8x16 | 1024x768 | 4/256k | | A000 | Trident TVGA 8900 |
| | = G | 128x48 | 8x16 | 1024x768 | 256/256k | | A000 | WD90C |
| | = G | 128x48 | 8x16 | 1024x768 | 256/256k | | A000 | Diamond Speedstar 24X |
| | = G | 128x48 | 8x16 | 1024x768 | 256/256k | | A000 | Cirrus CL-GD5420/ 5422/5426 |
| 61h | = T | 132x29 | 8x12 | | 16/64 | 8 | B800 | Quadram Ultra VGA |
| | = T | 132x29 | 8x8 | | 16 | | | Genoa 6400 |
| | = T | 132x29 | 8x8 | | 16 | | | Genoa SuperEGA BIOS 3.0+ |
| | = T | 132x50 | | | | | | Cirrus 5320 chipset |
| | = G | | | ?x400 | | | | Corona/Cordata BIOS 4.10+ |
| | = G | 80x25 | 8x16 | 640x400 | 256 | | A000 | ATI VGA Wonder, VGA Wonder+ |
| | = G | 80x25 | 8x16 | 640x400 | 256 | | A000 | ATI Ultra 8514A, ATI XL |
| | = G | 80x30 | 8x16 | 640x480 | 256 | 1 | A000 | Ahead A, Ahead B (512K) |
| | = G | | | 720x540 | | | | VEGA VGA |
| | = G | | | 720x540 | 16 | | | Tatung VGA |
| | = G | | | 720x540 | 16 | | | Video7 V-RAM VGA |
| | = G | 96x64 | 8x16 | 768x1024 | 16/256k | | A000 | Trident TVGA 88/8900 512K |
| | = G | 128x48 | 8x16 | 1024x768 | 256 | | A000 | NCR 77C22 [1,9] |
| 62h | = T | 132x32 | 8x11 | | 16/64 | 6 | B800 | Quadram Ultra VGA |
| | = T | 132x32 | 8x12 | | 16 | | | Genoa 6400 |
| | = T | 132x32 | 8x11 | | 16 | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | | | 640x450 | 16 | | | Cirrus 510/520 chipset |

| | text/grph | text resol | pixel box | pixel resoltn | colors | disply pages | scrn addr | system |
|---|---|---|---|---|---|---|---|---|
| | = G | 80x30 | 8x16 | 640x480 | 256 | | A000 | ATI VGA Wonder, VGA Wonder+ |
| | = G | 80x30 | 8x16 | 640x480 | 256 | | A000 | ATI Ultra 8514A, ATI XL |
| | = G | 80x30 | 8x16 | 640x480 | 32K | | A000 | WD90C |
| | = G | 80x30 | 8x16 | 640x480 | 32K | | A000 | Diamond Speedstar 24X |
| | = G | | | 800x600 | | | | VEGA VGA |
| | = G | | | 800x600 | 16 | | | Tatung VGA |
| | = G | | | 800x600 | 16 | | | Video7 V-RAM VGA |
| | = G | 100x75 | 8x8 | 800x600 | 256 | 1 | A000 | Ahead A, Ahead B (512K) |
| | = G | 128x48 | 8x16 | 1024x768 | 256/256k | | A000 | Trident TVGA 8900, Zymos |
| | = G | 128x48 | 8x16 | 1024x768 | 256 | | A000 | NCR 77C22 [9] |
| 63h | = T | 132x44 | 8x8 | | 16/64 | 5 | B800 | Quadram Ultra VGA |
| | = T | 132x44 | 8x8 | | 16 | | | Genoa 6400 |
| | = T | 132x44 | 8x8 | | 16 | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | | | 720x540 | 16 | | | MORSE VGA |
| | = G | | | 720x540 | 16 | | | Cirrus 510/520 chipset |
| | = G | 100x42 | 8x14 | 800x600 | 256 | | A000 | ATI VGA Wonder, VGA Wonder+ |
| | = G | 100x42 | 8x14 | 800x600 | 256 | | A000 | ATI Ultra 8514A, ATI XL |
| | = G | | | 800x600 | 32K | | A000 | WD90C |
| | = G | | | 800x600 | 32K | | A000 | Diamond Speedstar 24X |
| | = G | 128x48 | 7x16 | 1024x768 | 256 | 1 | A000 | Ahead B (1MB) |
| | = G | | | 1024x768 | | 2 | | Video7 V-RAM VGA |
| 64h | = T | 132x60 | 8x8 | | 16 | | | Genoa 6400 |
| | = G | | | 640x480 | 64K | | A000 | Cirrus CL-GD 5422/5426 |
| | = G | | | 800x600 | 16 | | | MORSE VGA |
| | = G | | | 800x600 | 16 | | | Cirrus 510/520 chipset |
| | = G | | | 800x600 | | | | SAMPO-Mira VGA |
| | = G | | | 1024x768 | 4 | | | Video7 V-RAM VGA |
| | = G | 128x48 | 8x16 | 1024x768 | 256 | | A000 | ATI VGA Wonder Plus, ATI XL |
| | = G | 160x64 | 8x16 | 1280x1024 | 16/256k | | A000 | WD90C [1] |
| | = G | 160x64 | 8x16 | 1280x1024 | 16/256k | | A000 | Diamond Speedstar 24X [1] |
| 65h | = G | | | 800x600 | 64K | | A000 | Cirrus CL-GD 5422/5426 |
| | = G | | | 1024x768 | 16 | | | Video7 V-RAM VGA |
| | = G | 128x48 | 8x16 | 1024x768 | 16 | | A000 | ATI VGA Wonder |
| 66h | = T | 80x50 | 8x8 | 640x400 | 16/256k | | B800 | WD90C |
| | = T | 80x50 | 8x8 | | 16 | | B800 | Diamond Speedstar 24X |
| | = G | | | 640x400 | 256 | | | Tatung VGA |
| | = G | | | 640x400 | 256 | | | Video7 V-RAM VGA |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| | = G | | | 640x480 | 32K | | A000 | Cirrus CL-GD 5422/5426 |
| 67h | = T | 80x43 | 8x8 | 640x344 | 16/256k | | B800 | WD90C |
| | = T | 80x43 | 8x8 | | 16 | | B800 | Diamond Speedstar 24X |
| | = G | | | 640x480 | 256 | | | Video7 V-RAM VGA |
| | = G | | | 800x600 | 32K | | A000 | Cirrus CL-GD 5422/5426 |
| | = G | 128x48 | 8x16 | 1024x768 | | 4 | A000 | ATI VGA Wonder |
| | = G | 160x64 | 8x16 | 1280x1024 | 16 | | A000 | NCR 77C22 [1,9] |
| 69h | = T | 132x50 | 8x8 | 1056x400 | 16/256k | | B800 | WD90C |
| | = T | 132x50 | 8x8 | | 16 | | B800 | Diamond Speedstar 24X |
| | = G | | | 720x540 | 256 | | | Video7 V-RAM VGA |
| 6Ah | = G | | | 800x600 | 16 | | A000 | VESA standard interface |
| | = G | 100x75 | 8x8 | 800x600 | 16 | | | Genoa 6400 |
| | = G | 100x75 | 8x8 | 800x600 | 16 | | | Diamond Speedstar 24X |
| | = G | | | 800x600 | 16 | | A000 | Ahead A |
| | = G | 100x75 | 8x8 | 800x600 | 16 | 1 | A000 | Ahead B (VESA) (see 71h) |
| | = G | | | 800x600 | 16 | | | Zymos Poach, Hi Res 512 |
| | = G | | | 800x600 | 16 | | | Epson LT-386SX in CRT Mode |
| | = G | | | 800x600 | 16 | | | Compuadd 316SL in CRT Mode |
| | = G | 100x37 | 8x16 | 800x600 | 16/256k | | A000 | Cirrus CL-GD 5420/5422/5426 |
| | = G | 100x42 | 8x14 | 800x600 | | | A000 | ATI VGA Wonder (undoc) |
| | = G | 160x64 | 8x16 | 1280x1024 | 256 | | A000 | NCR 77C22 [1,9] |
| 6Bh | = T | 100x37 | 8x16 | | 16 | | | Genoa 6400 |
| | = T | 100x37 | 8x16 | | | | | NEL Electronics BIOS |
| 6Ch | = G | 80x30 | 8x16 | 640x480 | 16M | | A000 | Trident 8900CL/ BIOS C04 |
| | = G | 100x75 | 8x8 | 800x600 | 256 | | | Genoa 6400 |
| | = G | 160x60 | 8x16 | 1280x960 | 16/256k | | A000 | WD90C [1] |
| | = G | 160x60 | 8x16 | 1280x960 | 16/256k | | A000 | Diamond Speedstar 24X [1] |
| | = G | 160x64 | 8x16 | 1280x1024 | 16/256k | | A000 | Cirrus CL-GD 5422/5426 [1] |
| 6Dh | = G | 160x64 | 8x16 | 1280x1024 | 256/256k | | A000 | Cirrus CL-GD 5422/5426 [1] |
| 6Eh | = G | 40x25 | 8x8 | 320x200 | 64K | | A000 | Cirrus CL-GD 5422/5426 |
| 6Fh | = G | 40x25 | 8x8 | 320x200 | 16M | | A000 | Cirrus CL-GD 5422/5426 |
| 70h | = | extended mode set (see Function 0070h) | | | | | | Everex Micro Enhancer EGA |
| | = T | 40x25 | 8x8 | | 16 | 8 | B800 | Quadram (CGA double scan) |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| | = T | 40x25 | 8x8 | (CGA dblscan) | | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | | | 360x480 | 256 | | | Cirrus 510/520/5320 chips |
| | = G | 90x28 | 8x14 | 720x392 | 16 | 1 | A000 | Ahead B |
| | = G | | | 800x600 | 16 | | | C&T chipset, Cardinal |
| | = G | | | 1024x480 | 256 | | | Trident 8900C BIOS C3.0 |
| 71h | = T | 80x25 | 8x8 | | 16 | 8 | B800 | Quadram (CGA double scan) |
| | = T | 80x25 | 8x8 | (CGA dblscan) | | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | | | 528x400 | 256 | | | Cirrus 510/520 chipset |
| | = G | 80x30 | 8x16 | 640x480 | 16M | | A000 | Cirrus CL-GD 5422/5426 |
| | = G | 100x35 | 8x16 | 800x600 | 16of64 | | A000 | NSI Smart EGA+ |
| | = G | 100x75 | 8x8 | 800x600 | 16 | 1 | A000 | Ahead B (same as 6Ah) |
| | = G | | | 960x720 | 16 | | | C&T chipset, Cardinal |
| | = G | | | 1024x480 | 256 | | | Trident 8900C BIOS C3.0 |
| 72h | = T | 80x60 | 8x8 | | 16 | | B800 | Quadram Ultra VGA |
| | = T | 80x60 | 8x8 | | 16 | | B800 | Genoa 6400 |
| | = T | 80x60 | 8x8 | | 16 | | B800 | Genoa SuperEGA BIOS 3.0+ |
| | = G | | | 528x480 | 256 | | | Cirrus 510/520 chipset |
| | = G | | | 640x480 | 32768 | | A000 | ATI |
| | = G | | | 640x480 | 16M | | A000 | WD90C |
| | = G | | | 640x480 | 16M | | A000 | Diamond Speedstar 24X |
| | = G | | | 1024x768 | 16 | | | C&T chipset, Cardinal |
| 73h | = G | 80x60 | 8x8 | 640x480 | 16 | | A000 | Quadram Ultra VGA |
| | = G | 80x60 | 8x8 | 640x480 | 16 | | | Genoa 6400 |
| | = G | 80x60 | 8x8 | 640x480 | 16 | | | Genoa SuperEGA BIOS 3.0+ |
| 74h | = T | 80x66 | 8x8 | | 16 | | B800 | Quadram Ultra VGA |
| | = T | 80x66 | 8x8 | | 16 | | B800 | Genoa 6400 |
| | = T | 80x66 | 8x8 | | 16 | | B800 | Genoa SuperEGA BIOS 3.0+ |
| | = G | | | 640x400 | 2 | | B800 | Toshiba 3100 AT&T mode |
| | = G | 80x30 | 8x16 | 640x480 | 32K | | A000 | Trident 8900C/ BIOS C03 |
| | = G | 128x48 | 8x16 | 1024x768 | 16 | 1 | A000 | Ahead A, Ahead B (512K) |
| | = G | | | 1024x768 | 64K | | A000 | Cirrus CL-GD 5422/5426 [1] |
| 75h | = G | 80x30 | 8x16 | 640x480 | 64K | | A000 | Trident 8900C/ BIOS C03 |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| | = G | 80x66 | | 640x528 | *16* | | A000 | Quadram Ultra VGA |
| | = G | 80x66 | | 640x528 | 16 | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | 128x48 | 8x16 | 1024x768 | 4 | 1 | A000 | Ahead B |
| 76h | = T | 94x29 | 8x14 | | 16 | | B800 | Quadram Ultra VGA |
| | = T | 94x29 | 8x14 | | | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | 100x75 | 8x8 | 800x600 | 32K | | A000 | Trident 8900C/ BIOS C03 |
| | = G | 128x48 | 8x16 | 1024x768 | 2 | 1 | A000 | Ahead B |
| 77h | = G | 94x29 | | 752x410 | *16* | | A000 | Quadram Ultra VGA |
| | = G | 94x29 | | 752x410 | 16 | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | 100x75 | 8x8 | 800x600 | 64K | | A000 | Trident 8900C/ BIOS C03 |
| 78h | = T | 100x37 | 8x16 | | 16 | | | Genoa 6400 |
| | = T | 100x75 | 8x8 | | 16 | | B800 | Quadram Ultra VGA |
| | = T | 100x75 | 8x8 | | | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | | | 640x400 | 256 | | | STB VGA/EM-16 Plus |
| | = G | | | 640x400 | 256 | | | Cardinal, C&T chipset |
| | = G | | | 640x400 | 256 | | | Cirrus 5320 chipset |
| 79h | = G | | | 640x480 | 256 | | | Cardinal, C&T chipset |
| | = G | 100x75 | | 800x600 | *16* | | A000 | Quadram Ultra VGA |
| | = G | 100x75 | 8x8 | 800x600 | 16 | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | 100x75 | 8x8 | 800x600 | 16 | | | Genoa 6400 |
| 7Ah | = T | 114x60 | 8x8 | | 16 | | B800 | Quadram Ultra VGA |
| | = T | 114x60 | 8x8 | | | | | Genoa SuperEGA BIOS 3.0+ |
| | = G | | | 720x540 | 256 | | | C&T chipset, Cardinal |
| 7Bh | = G | | | 800x600 | 256 | | | C&T chipset, Cardinal |
| | = G | 114x60 | | 912x480 | *16* | | A000 | Quadram Ultra VGA |
| | = G | | | 912x480 | 16 | | | Genoa SuperEGA BIOS 3.0+ |
| 7Ch | = G | | | 512x512 | 16 | | | Genoa |
| 7Dh | = G | 64x32 | 8x16 | 512x512 | 256 | | | Genoa |
| 7Eh | = | special mode set (see Function 007Eh) | | | | | | Paradise VGA, AT&T VDC600 |
| | = G | 80x25 | 8x16 | 640x400 | 256 | | | Genoa 6400 |
| 7Fh | = | special function set (see Function 007Fh) | | | | | | Paradise VGA, AT&T VDC600 |
| | = G | 128x48 | 8x16 | 1024x768 | 4 | | | Genoa 6400 |
| 82h | = T | 80x25 | | | B&W | | | AT&T VDC overlay mode [6] |
| 83h | = T | 80x25 | | | | | | AT&T VDC overlay mode [6] |

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disply pages* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| 86h | = G | | | 640x200 | B&W | | | AT&T VDC overlay mode [6] |
| 88h | = G | 90x43 | 8x8 | 720x352 | mono | | B000 | Hercules + MSHERC.COM |
| C0h | = G | | | 640x400 | 2/prog pallet | | | AT&T VDC overlay mode [6] |
| C4h | = | disable output | | | | | | AT&T VDC overlay mode [6] |
| D0h | = G | | | 640x400 | 2 | | B800 | DEC VAXmate AT&T mode |

**See also:** Tables 8-19, 8-27, 8-37, 9-4, and 9-5

**Notes:**

* interlaced only
* For the ATI EGA Wonder, mode 08h is only valid if SMS.COM is loaded resident. SMS maps mode 08h to mode 27h if the byte at location 0040:0063 is 0B4h, otherwise to mode 23h, thus selecting the appropriate (monochrome or color) 132x25 character mode. For the ATI VGA Wonder, mode 08h is the same, and only valid if VCONFIG is loaded resident.
* early XGA boards support 132-column text but do not have this BIOS mode
* DESQview intercepts calls to change into these two modes (21h is page 0, 22h is page 1) even if there is no Hercules graphics board installed.
* The ATI BIOS v4-1.00 has a text-scrolling bug in this mode.
* For AT&T VDC overlay modes, BL contains the DEB mode, which may be 06h, 40h, or 44h.
* BIOS text support is broken in this undocumented mode; scrolling moves only about 1/3 of the screen (and does even that portion incorrectly), while screen clears only clear about 3/4.
* The Oak OTI-037/067/077 modes are present in the Oak VGA BIOS, which OEMs may choose to use only partially or not at all; thus, not all Oak boards support all " Oak " modes listed here.
* This card uses the full 128K A000h-BFFFh range for the video buffer, precluding the use of a monochrome adapter in the same system.
* Mode 17h is supported by the Tseng ET4000 BIOS 8.01X dated 09/14/90, but not v8.01X dated 02/28/92; mode 21h is supported by the 02/28/92 version but not the 09/14/90 version.
* HERKULES simulates a 90x45 text mode in Hercules graphics mode; the installation check for HERKULES.COM is the signature string " Herkules " two bytes beyond the INT 10h handler.
* The Realtek RTVGA BIOS v3.C10 crashes when attempting to switch into modes 21h or 27h; this version of the BIOS also sets the BIOS data area incorrectly for extended text modes, resulting in scrolling after only 24 lines (the VMODE.EXE utility does set the data area correctly).

---

## INT 10h Function 01h
### *SET TEXT-MODE CURSOR SHAPE*

**Purpose:** Specify the starting and ending scan lines to be occupied by the hardware cursor in text modes.

**Registers at call:**

AH = 01h
CH = bit 7 should be zero
    bits 6,5 cursor blink
        (00=normal, 01=invisible, 10=erratic, 11=slow)

(00=normal, other=invisible on EGA/VGA)
bits 4-0 top scan line containing cursor
CL = bottom scan line containing cursor (bits 0-4)

**Return Registers:**
none

**Details:** This function is buggy on EGA systems--the BIOS remaps the cursor shape in 43 line modes, but returns the unmapped cursor shape.

UltraVision scales the size to the current font height by assuming 14-line monochrome and 8-line color fonts; this call is not valid if cursor emulation has been disabled.

Applications which wish to change the cursor by programming the hardware directly on EGA or above should call INT 10h Function 1130h or read 0040h:0085h first to determine the current font height

**BUG:** AMI 386 BIOS and AST Premier 386 BIOS will lock up the system if AL is not equal to the current video mode.

**See also:** Function 03h, Function 12h Subfunction 34h, UltraVision Function CD05h

---

## INT 10h Function 02h
### *SET CURSOR POSITION*

**Registers at call:**
AH = 02h
BH = page number
  0-3 in modes 2&3
  0-7 in modes 0&1
  0 in graphics modes
DH = row (00h is top)
DL = column (00h is left)

**Return Registers:**
none

**See also:** Functions 03h and 05h, HP 95LX INT 60h Function 030Bh (Chapter 52)

---

## INT 10h Function 03h
### *GET CURSOR POSITION AND SIZE*

**Registers at call:**
AH = 03h
BH = page number
  0-3 in modes 2&3
  0-7 in modes 0&1
  0 in graphics modes

**Return Registers:**
AX = 0000h (Phoenix BIOS)
CH = start scan line
CL = end scan line
DH = row (00h is top)
DL = column (00h is left)

**Details:** A separate cursor is maintained for each of up to 8 display pages.

Many ROM BIOSes incorrectly return the default size for a color display (start 06h, end 07h) when a monochrome display is attached.

**See also:** Functions 01h and 02h, Function 12h Subfunction 34h

---

## INT 10h Function 04h
### *READ LIGHT PEN POSITION*

*all except VGA*

**Purpose:** Determine the current position and status of the light pen (if present).

**Registers at call:**
AH = 04h

**Return Registers:**
AH = light pen trigger flag
  00h not down/triggered
  01h down/triggered
    DH,DL = row,column of character light pen is on
    CH = pixel row (graphics modes 04h-06h)
    CX = pixel row (graphics modes with >200 rows)
    BX = pixel column

**Details:** On a CGA, returned column numbers are always multiples of 2 (320- column modes) or 4 (640-column modes).

The returned row numbers are only accurate to two lines.

### INT 10h Function 05h
### *SELECT ACTIVE DISPLAY PAGE*

**Purpose:** Specify which of possibly multiple display pages will be visible.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 05h | none |
| AL = new page number (00h to number of pages - 1) (see Function 00h) | |

**Details:** To determine whether the requested page actually exists, use Function 0Fh to query the current page after making this call.

**Conflicts:** PCjr Function 05h below

**See also:** Function 0Fh, GRAFIX Functions 43h and 45h (below)

### INT 10h Function 05h, Subfunctions 80h to 83h
*PCjr*
### *MANIPULATE CRT/CPU PAGE REGISTERS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 05h | |
| AL = subfunction | |
| 80h read CRT and CPU page registers | BH = CRT page register<br>BL = CPU page register |
| 81h set CPU page register<br>BL = CPU page | |
| 82h set CRT page register<br>BH = CRT page | |
| 83h set both CPU and CRT page registers<br>BL = CPU page<br>BH = CRT page | |

**Details:** The CPU page determines which 16K block of the first 128K of physical memory will be mapped at B800h by the hardware. The CRT page determines the start address of the memory used by the video controller.

### INT 10h Function 06h
### *SCROLL UP WINDOW*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 06h | none |
| AL = number of lines by which to scroll up (00h = clear entire window) | |
| BH = attribute used to write blank lines at bottom of window | |
| CH,CL = row,column of window's upper left corner | |
| DH,DL = row,column of window's lower right corner | |

**Details:** This function affects only the currently active page (see Function 05h)

**Warning:** Some implementations have a bug which destroys BP.

**See also:** Function 07h, Tandy Functions 72h and 73h (below), SOLLEX Function 7F07h (Chapter 9), PC Thuis INT 50h Function 0014h (Chapter 66)

### INT 10h Function 07h
### *SCROLL DOWN WINDOW*

**Registers at call:**

AH = 07h

AL = number of lines by which to scroll down

(00h=clear entire window)
BH = attribute used to write blank lines at top of window
CH,CL = row,column of window's upper left corner
DH,DL = row,column of window's lower right corner

**Return Registers:**
none

**Details:** This function affects only the currently active page (see Function 05h)
**Warning:** some implementations have a bug which destroys BP.
**See also:** Function 06h, Tandy Functions 72h and 73h,PC Thuis INT 50h Function 0014h (Chapter 66)

---

### INT 10h Function 08h
### *READ CHARACTER AND ATTRIBUTE AT CURSOR POSITION*

**Registers at call:**
AH = 08h
BH = page number (00h to number of pages - 1) (see Function 00h)

**Return Registers:**
AH = charater's attribute (see Table 8-3)
AL = character

**Details:** For monochrome displays, a foreground of 1 with background 0 is underlined.

The blink bit may be reprogrammed to enable intense background colors using Function 1003h or by programming the CRT controller.

The foreground intensity bit (3) can be programmed to switch between character sets A and B on EGA and VGA cards, thus enabling 512 simultaneous characters on screen. In this case the bit's usual function (intensity) is regularly turned off.

**See also:** Function 09h, Function 1003h, AX PC Function 5001h (Chapter 6)

**8-3 Bitfields for character's attribute:**

| | | | |
|---|---|---|---|
| bit 7 | blink | | |
| bits 6-4 | background color | | |
| | | 000 black | 100 red |
| | | 001 blue | 101 magenta |
| | | 010 green | 110 brown |
| | | 011 cyan | 111 white |
| bits 3-0 | foreground color | | |
| | | 0000 black | 1000 dark gray |
| | | 0001 blue | 1001 light blue |
| | | 0010 green | 1010 light green |
| | | 0011 cyan | 1011 light cyan |
| | | 0100 red | 1100 light red |
| | | 0101 magenta | 1101 light magenta |
| | | 0110 brown | 1110 yellow |
| | | 0111 light gray | 1111 white |

---

### INT 10h Function 09h
### *WRITE CHARACTER AND ATTRIBUTE AT CURSOR POSITION*

**Registers at call:**
AH = 09h
AL = character to display
BH = page number (00h to number of pages - 1) (see Function 00h)
BL = attribute (text mode) or color (graphics mode)
  if bit 7 set in graphics mode, character is xor'ed onto screen
CX = number of times to write character

**Return Registers:**
none

**Details:** All characters are displayed, including CR, LF, and BS.

The replication count in CX may produce an unpredictable result in graphics modes if it is greater than the number of positions remaining in the current row.

**See also:** Functions 08h and 0Ah, INT 1Fh, INT 43h, INT 44h, GRAFIX Function 4Bh (below), FLASHUP INT 17h Function 60h (below)

---

### INT 10h Function 0Ah
### *WRITE CHARACTER ONLY AT CURSOR POSITION*

**Registers at call:**
AH = 0Ah
AL = character to display
BH = page number (00h to number of pages - 1) (see Function 00h)
BL = attribute (PCjr only) or color (graphics mode)
    if bit 7 set in graphics mode, character is XORed onto screen
CX = number of times to write character

**Return Registers:**
none

**Details:** All characters are displayed, including CR, LF, and BS.

The replication count in CX may produce an unpredictable result in graphics modes if it is greater than the number of positions remaining in the current row.

**See also:** Functions 08h and 09h, INT 1Fh, INT 43h, INT 44h, GRAFIX Function 4Bh (below), FLASHUP INT 17h Function 60h (below)

---

### INT 10h Function 0Bh, Subfunction 00h
### *SET BACKGROUND/BORDER COLOR*

**Registers at call:**
AH = 0Bh
BH = 00h
BL = background/border color (border only in text modes)

**Return Registers:**
none

**See also:** Function 0Bh Subfunction 01h

---

### INT 10h Function 0Bh, Subfunction 01h
### *SET PALETTE*

**Registers at call:**
AH = 0BH
BH = 01h
BL = palette ID
    00h background, green, red, and brown/yellow
    01h background, cyan, magenta, and white

**Return Registers:**
none

**See also:** Function 0Bh Subfunction 00h

---

### INT 10h Function 0Ch
### *WRITE GRAPHICS PIXEL*

**Purpose:** Set a single pixel on the display in graphics modes.

**Registers at call:**
AH = 0Ch
BH = page number
AL = pixel color (if bit 7 set, value is xor'ed onto screen)
CX = column
DX = row

**Return Registers:**
none

**Details:** This function is valid only in graphics modes.

BH is ignored if the current video mode supports only one page.

**See also:** Function 0Dh, GRAFIX Function 46h (below)

### INT 10h Function 0Dh
### *READ GRAPHICS PIXEL*

**Purpose:** Determine the current color of the specified pixel in graphics modes.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 0Dh<br>BH = page number<br>CX = column<br>DX = row | AL = pixel color |

**Details:** This function is valid only in graphics modes.

BH is ignored if the current video mode supports only one page.

**See also:** Function 0Ch, GRAFIX Function 47h (below)

### INT 10h Function 0Eh
### *TELETYPE OUTPUT*

**Purpose:** Display a character on the screen, advancing the cursor and scrolling the screen as necessary.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 0Eh<br>AL = character to write<br>BH = page number<br>BL = foreground color (graphics modes only) | none |

**Details:** Characters 07h (BEL), 08h (BS), 0Ah (LF), and 0Dh (CR) are interpreted and do the expected things.

IBM PC ROMs dated 4/24/81 and 10/19/81 require that BH be the same as the current active page.

**See also:** Functions 02h and 0Ah

### INT 10h Function 0Fh
### *GET CURRENT VIDEO MODE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 0Fh | AH = number of character columns<br>AL = display mode (see Table 8-2)<br>BH = active page (see Function 05h) |

**Details:** If the mode was set with bit 7 set ( " no blanking " ), the returned mode will also have bit 7 set.

EGA, VGA, and UltraVision return either AL=03h (color) or AL=07h (monochrome) in all extended-row text modes.

**Conflicts:** VUIMAGE (below), Frieze (Chapter 15)

**See also:** Functions 00h, 05h, and 1130h; UltraVision Function CD04h (below); Tseng Function 10F2h Subfunction 00h (Chapter 9)

### INT 10h Function 1000h
### *SET SINGLE PALETTE REGISTER*

*PCjr, EGA, MCGA, VGA*

**Registers at call:**

AX = 1000h

BL = palette register number (00h-0Fh)
    or attribute register number (undocumented):
    10h attribute mode control register (should let BIOS control this)
    11h overscan color register (see also Function 1001h)
    12h color plane enable register (bits 3-0 enable corresponding text attribute bit)
    13h horizontal PEL panning register

14h color select register
BH = color or attribute register value

**Return Registers:**
none

**Details:** On MCGA, only BX = 0712h is supported.

Under UltraVision, the palette locking status (see Function CD01h) determines the outcome.

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)

**See also:** Functions 1002h and 1007h, UltraVision Function CD01h (below)

---

## INT 10h Function 1001h
*PCjr, EGA, VGA*

### *SET BORDER (OVERSCAN) COLOR*

**Registers at call:**
AX = 1001h
BH = border color (00h-3Fh)

**Return Registers:**
none

**Details:** Under UltraVision, the palette locking status (see Function CD01h) determines the outcome.

**BUG:** The original IBM VGA BIOS incorrectly updates the parameter save area and places the border color at offset 11h of the palette table rather than offset 10h.

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)

**See also:** Functions 1002h and 1008h, UltraVision Function CD01h (below)

---

## INT 10h Function 1002h
*PCjr, EGA, VGA*

### *SET ALL PALETTE REGISTERS*

**Registers at call:**
AX = 1002h
ES:DX -> palette register list (see Table 8-4)

**Return Registers:**
none

**Details:** Under UltraVision, the palette locking status (see Function CD01h) determines the outcome.

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)

**See also:** Functions 1000h, 1001h, and 1009h; UltraVision Function CD01h (below)

**8-4 Format of palette register list:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 16 BYTEs | colors for palette registers 00h through 0Fh |
| 10h | BYTE | border color |

---

## INT 10h Function 1003h
*PCjr, Tandy 1000, EGA, VGA, MCGA*

### *TOGGLE INTENSITY/BLINKING BIT*

**Registers at call:**
AX = 1003h
BL = new state
    00h background intensity enabled
    01h blink enabled
BH = 00h to avoid problems on some adapters

**Return Registers:**
none

**Details:** Although there is no function to get the current status on adapters prior to the VGA, bit 5 of 0040h:0065h indicates the state; on the VGA, use Function 1Bh and check offset 2Dh of the returned data.

**BUG:** When configured for a monochrome display, the Boca Research Multi-EGA with ROM v M1.1 Type D has its screen disrupted if BH is not clear.

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)

**See also:** Functions 08h and 1Bh

---

## INT 10h Function 1007h

*VGA, UltraVision v2+*

### *GET INDIVIDUAL PALETTE REGISTER*

**Registers at call:**
AX = 1007h
BL = palette or attribute (undocumented) register number (see Function 1000h)

**Return Registers:**
BH = palette or attribute register value

**Details:** UltraVision v2+ supports this function even on color EGA systems in video modes 00h-03h, 10h, and 12h; direct programming of the palette registers will cause incorrect results because the EGA registers are write-only. To guard against older versions or unsupported video modes, programs which expect to use this function on EGA systems should set BH to FFh on entry.

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)

**See also:** Functions 1000h and 1009h

---

## INT 10h Function 1008h

*VGA, UltraVision v2+*

### *READ OVERSCAN (BORDER COLOR) REGISTER*

**Registers at call:**
AX = 1008h

**Return Registers:**
BH = border color (00h-3Fh)

**Details:** (see Function 1007h).

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)

**See also:** Function 1001h

---

## INT 10h Function 1009h

*VGA*

### *READ ALL PALETTE REGISTERS AND OVERSCAN REGISTER*

**Registers at call:**
AX = 1009h
ES:DX -> 17-byte buffer for palette register list (see Table 8-4)

**Return Registers:**
none

**Details:** UltraVision v2+ supports this function even on color EGA systems in video modes 00h-03h, 10h, and 12h; direct programming of the palette registers will cause incorrect results because the EGA registers are write-only. To guard against older versions or unsupported video modes, programs which expect to use this function on EGA systems should set the ES:DX buffer to FFh before calling.

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)

**See also:** Functions 1002h and 1007h, UltraVision Function CD02h (below)

---

## INT 10h Function 1010h

*VGA, MCGA*

### *SET INDIVIDUAL DAC REGISTER*

**Registers at call:**
AX = 1010h
BX = register number
CH = new value for green (0-63)
CL = new value for blue (0-63)
DH = new value for red (0-63)

**Return Registers:**
none

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)

**See also:** Functions 1012h and 1015h

---

## INT 10h Function 1012h

*VGA, MCGA*

### *SET BLOCK OF DAC REGISTERS*

**Registers at call:**
AX = 1012h
BX = starting color register
CX = number of registers to set
ES:DX -> table of 3*CX bytes where each 3-byte

group represents one byte each of red, green and blue (0-63)

**Return Registers:**
none

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)
**See also:** Functions 1010h and 1017h, FastGraph/Light INT 62h Function 00A5h (Chapter 52)

---

### INT 10h Function 1013h
### *SELECT VIDEO DAC COLOR PAGE*

*VGA*

**Registers at call:**
AX = 1013h
BL = subfunction
    00h select paging mode
        BH = 00h select 4 blocks of 64
        BH = 01h select 16 blocks of 16
    01h select page
        BH = page number (00h to 03h) or (00h to 0Fh)

**Return Registers:**
none

**Details:** This function is not valid in mode 13h (see also Function 00h).
**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)
**See also:** Function 101Ah

---

### INT 10h Function 1015h
### *READ INDIVIDUAL DAC REGISTER*

*VGA, MCGA*

**Registers at call:**
AX = 1015h
BL = palette register number

**Return Registers:**
DH = red value
CH = green value
CL = blue value

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)
**See also:** Functions 1010h and 1017h

---

### INT 10h Function 1017h
### *READ BLOCK OF DAC REGISTERS*

*VGA, MCGA*

**Registers at call:**
AX = 1017h
BX = starting palette register
CX = number of palette registers to read
ES:DX -> buffer (3 * CX bytes in size) (see also Function 1012h)

**Return Registers:**
buffer filled with CX red, green and blue triples

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)
**See also:** Functions 1012h and 1015h, FastGraph/Light INT 62h Function 00A4h (Chapter 52)

---

### INT 10h Function 1018h
### *SET PEL MASK*

*VGA, MCGA*
*(undocumented)*

**Registers at call:**
AX = 1018h
BL = new PEL value

**Return Registers:**
none

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)
**See also:** Function 1019h

---

### INT 10h Function 1019h
### *READ PEL MASK*

*VGA, MCGA*
*(undocumented)*

**Registers at call:**
AX = 1019h

**Return Registers:**
BL = value read

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)
**See also:** Function 1018h

---

## INT 10h Function 101Ah

*VGA*

### *GET VIDEO DAC COLOR-PAGE STATE*

**Registers at call:**
AX = 101Ah

**Return Registers:**
BL = paging mode
    00h four pages of 64
    01h sixteen pages of 16
BH = current page

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)
**See also:** Function 1013h

---

## INT 10h Function 101Bh

*VGA, MCGA*

### *PERFORM GRAY-SCALE SUMMING*

**Purpose:** Convert the RGB values of one or more palette registers such that the resulting values are grays with the same intensities as the original colors.

**Registers at call:**
AX = 101Bh
BX = starting palette register
CX = number of registers to convert

**Return Registers:**
none

**Conflicts:** BIOS Window Extension (below), Eagle PC2 BIOS (below)
**See also:** Function 12h Subfunction 33h

---

## INT 10h Function 11h, various subfunctions

*EGA, VGA, MCGA*

### *TEXT-MODE CHARACTER GENERATOR FUNCTIONS*

**Purpose:** Specify the fonts to be used in various circumstances. These functions will cause a mode set, completely resetting the video environment, but without clearing the video buffer.

**Registers at call:**
AH = 11h
AL = 00h and 10h: load user-specified patterns
    ES:BP -> user table
    CX = count of patterns to store
    DX = character offset into map 2 block
    BL = block to load in map 2
    BH = number of bytes per character pattern
AL = 01h and 11h: load ROM monochrome patterns (8 by 14)
    BL = block to load
AL = 02h and 12h: load ROM 8 by 8 double-dot patterns
    BL = block to load
AL = 03h: set block specifier
    BL = block specifier
        EGA/MCGA
        bits 0,1 = block selected by chars with attribute bit 3 = 0
        bits 2,3 = block selected by chars with attribute bit 3 = 1
        **VGA**
        bits 0,1,4 = block selected by attribute bit 3 = 0
        bits 2,3,5 = block selected by attribute bit 3 = 1
AL = 04h and 14h: load ROM 8x16 character set (VGA)
    BL = block to load

**Return Registers:**
none

**Details:** The routines called with AL=1xh are designed to be called only immediately after a mode set and are similar to the routines called with AL=0xh, except that:

- Page 0 must be active.
- Bytes/character is recalculated.

- Maximum character row is recalculated.
- CRT buffer length is recalculated.
- CRTC registers are reprogrammed as follows:
- R09 = bytes/char-1 (max scan line, mode 7 only)
- R0A = bytes/char-2 (cursor start)
- R0B = 00h (cursor end)
- R12 = ((rows+1)*(bytes/char))-1 (vertical display end)
- R14 = bytes/char (underline location) (*BUG: should be 1 less*).

The current block specifiers may be determined with INT 10h Function 1Bh, looking at offsets 2Bh and 2Ch of the returned data (VGA only).

**Conflicts:** BIOS Window Extension (below), Realtek RTVGA (below)

**See also:** Function 1Bh, UltraVision Function CD10h (below)

---

### INT 10h Function 1120h
*EGA, VGA, MCGA*

### *GRAPH-MODE CHARGEN - SET USER 8x8 GRAPHICS CHARACTERS*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1120h | none |
| ES:BP -> user table for INT 1Fh | |

**Details:** This function is meant to be called immediately after a mode set; results are unpredictable at other times.

**Conflicts:** BIOS Window Extension (below)

**See also:** Functions 1121h through 1125h, INT 1Fh, INT 43h

---

### INT 10h Function 1121h
*EGA, VGA, MCGA*

### *GRAPH-MODE CHARGEN - SET USER GRAPHICS CHARACTERS*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1121h | none |

ES:BP -> user table
CX = bytes per character
BL = row specifier
  00h user set
    DL = number of rows
  01h 14 rows
  02h 25 rows
  03h 43 rows

**Details:** This function is meant to be called immediately after a mode set; results are unpredictable at other times.

**Conflicts:** BIOS Window Extension (below)

**See also:** Functions 1120h, 1122h, 1123h, and 1124h; INT 1Fh; INT 43h; Compaq Function 1129h (below)

---

### INT 10h Function 1122h
*EGA, VGA, MCGA*

### *GRAPH-MODE CHARGEN - SET ROM 8x14 GRAPHICS CHARACTERS*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1122h | none |
| BL = row specifier (see Function 1121h) | |

**Details:** This function is meant to be called immediately after a mode set; results are unpredictable at other times.

UltraVision v2+ sets INT 43h to the appropriate font on this call.

**Conflicts:** BIOS Window Extension (below)

**See also:** Functions 1111h, 1120h, 1121h, 1123h, and 1124h; INT 1Fh; INT 43h; Compaq Function 1129h (below)

### INT 10h Function 1123h

*EGA, VGA, MCGA*

### *GRAPH-MODE CHARGEN - SET ROM 8x8 DOUBLE-DOT CHARACTERS*

**Registers at call:**
AX = 1123h
BL = row specifier (see Function 1121h)

**Return Registers:**
none

**Details:** This function is meant to be called immediately after a mode set; results are unpredictable at other times.

UltraVision v2+ sets INT 43h to the appropriate font on this call.

**Conflicts:** BIOS Window Extension (below)

**See also:** Functions 1112h, 1120h, 1121h, 1122h, and 1124h; INT 1Fh; INT 43h; Compaq Function 1129h

### INT 10h Function 1124h

*VGA, MCGA*

### *GRAPH-MODE CHARGEN - LOAD 8x16 GRAPHICS CHARACTERS*

**Registers at call:**
AX = 1124h
BL = row specifier (see Function 1121h)

**Return Registers:**
none

**Details:** This function is meant to be called immediately after a mode set; results are unpredictable at other times.

UltraVision v2+ sets INT 43h to the appropriate font on this call.

**Conflicts:** BIOS Window Extension (below)

**See also:** Functions 1120h through 1123h, INT 1Fh, INT 43h, Compaq Function 1129h

### INT 10h Function 1130h

*EGA, MCGA, VGA*

### *GET FONT INFORMATION*

**Registers at call:**
AX = 1130h
BH = pointer specifier
- 00h INT 1Fh pointer
- 01h INT 43h pointer
- 02h ROM 8x14 character font pointer
- 03h ROM 8x8 double dot font pointer
- 04h ROM 8x8 double dot font (high 128 characters)
- 05h ROM alpha alternate (9 by 14) pointer (EGA,VGA)
- 06h ROM 8x16 font (MCGA, VGA)
- 07h ROM alternate 9x16 font (VGA only)
- 11h (UltraVision v2+) 8x20 font (VGA) or 8x19 font (autosync EGA)
- 12h (UltraVision v2+) 8x10 font (VGA) or 8x11 font (autosync EGA)

**Return Registers:**
ES:BP = specified pointer
CX = bytes/character of on-screen font
(not the requested font!)
DL = maximum character row
(character rows - 1) for VGA; some
EGA return character rows instead

**Details:** For UltraVision v2+, the 9xN alternate fonts follow the corresponding 8xN font at ES:BP+256N.

**Conflicts:** BIOS Window Extension (below), M10_SCR (below)

**See also:** Functions 1100h and 1120h, INT 1Fh, INT 43h

**8-5 Format of alternate font table [array]:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | character to be replaced (00h = end of table) |
| 01h | N BYTEs | graphics data for character, one byte per scan line |

### INT 10h Function 12h, Subfunction 10h

*EGA, VGA, MCGA*

### *ALTERNATE FUNCTION SELECT - GET EGA INFO*

**Registers at call:**
AH = 12h
BL = 10h

**Return Registers:**
BH = 00h color mode in effect (I/O port 3Dxh)
01h mono mode in effect (I/O port 3Bxh)

BL = installed memory (00h = 64K, 01h = 128K, 02h = 192K, 03h = 256K)
CH = feature bits
CL = switch settings
  bit 0: switch 1 OFF
  bit 1: switch 2 OFF
  bit 2: switch 3 OFF
  bit 3: switch 4 OFF
  bits 4-7: unused

**Details:** One possible check for the presence of an EGA or later display card is to call this function with BH=FFh; if not present, BH will be unchanged on return.
**Conflicts:** BIOS Window Extension (below)

---

### INT 10h Function 12h, Subfunction 20h *EGA, VGA, MCGA*
### *ALTERNATE FUNCTION SELECT - SELECT ALTERNATE PRTSC*

**Purpose:** Installs a PrintScreen routine from the video card's BIOS to replace the default PrtSc handler from the ROM BIOS, which usually does not understand screen heights other than 25 lines.

| Registers at call: | Return Registers: |
|---|---|
| AH = 12h<br>BL = 20h | none |

**Details:** Some adapters disable print-screen instead of enhancing it.
**Conflicts:** BIOS Window Extension (below)
**See also:** BIOS INT 05h (Chapter 4)

---

### INT 10h Function 12h, Subfunction 30h *VGA*
### *ALTERNATE FUNCTION SELECT - SELECT VERTICAL RESOLUTION*

**Purpose:** Specify the number of scan lines used to display text modes.

| Registers at call: | Return Registers: |
|---|---|
| AH = 12h<br>BL = 30h<br>AL = vertical resolution<br>00h 200 scan lines<br>01h 350 scan lines<br>02h 400 scan lines | AL = 12h if function supported |

**Details:** The specified resolution will take effect on the next mode set.
**Conflicts:** BIOS Window Extension (below)
**See also:** Function 00h

---

### INT 10h Function 12h, Subfunction 31h *VGA, MCGA*
### *ALTERNATE FUNCTION SELECT - PALETTE LOADING*

**Purpose:** Specify whether a default palette should be loaded when the display mode is set.

| Registers at call: | Return Registers: |
|---|---|
| AH = 12h<br>BL = 31h<br>AL = new state (00h enabled, 01h disabled) | AL = 12h if function supported |

**Conflicts:** BIOS Window Extension (below)
**See also:** Function 00h

---

### INT 10h Function 12h, Subfunction 32h *VGA, MCGA*
### *ALTERNATE FUNCTION SELECT - VIDEO ADDRESSING*

**Purpose:** Specify whether the CPU should have access to video memory and the display adapters I/O registers.

**Registers at call:**
AH = 12h
BL = 32h
AL = new state (00h enabled, 01h disabled)

**Return Registers:**
AL = 12h if function supported

**Conflicts:** BIOS Window Extension (below)

---

### INT 10h Function 12h, Subfunction 33h

*VGA, MCGA*

***ALTERNATE FUNCTION SELECT - GRAY-SCALE SUMMING***

**Purpose:** Specify whether or not colors should be converted to gray scale when palette or color registers are loaded.

**Registers at call:**
AH = 12h
BL = 33h
AL = new state (00h enabled, 01h disabled)

**Return Registers:**
AL = 12h if function supported

**Conflicts:** BIOS Window Extension (below)
**See also:** Function 101Bh, Compaq Functions BF06h and BF08h (below)

---

### INT 10h Function 12h, Subfunction 34h

*VGA*

***ALTERNATE FUNCTION SELECT - CURSOR EMULATION***

**Purpose:** Specify whether the BIOS should automatically remap cursor start/end according to the current character height in text modes.

**Registers at call:**
AH = 12h
BL = 34h
AL = new state (00h enabled, 01h disabled)

**Return Registers:**
AL = 12h if function supported

**Conflicts:** BIOS Window Extension (below)
**See also:** Functions 01h and 03h

---

### INT 10h Function 12h, Subfunction 35h

*PS/2*

***ALTERNATE FUNCTION SELECT - DISPLAY-SWITCH INTERFACE***

**Purpose:** Switch between two video adapters which may otherwise have address conflicts.

**Registers at call:**
AH = 12h
BL = 35h
AL = subfunction
    00h initial adapter video off
    01h initial planar video on
    02h switch active video off
    03h switch inactive video on
    80h **undocumented** set system board video active flag
ES:DX -> buffer for 128 byte save area (if AL = 0, 2 or 3)

**Return Registers:**
AL = 12h if function supported

**Conflicts:** BIOS Window Extension (below)

---

### INT 10h Function 12h, Subfunction 36h

*VGA, MCGA, PS/2*

***ALTERNATE FUNCTION SELECT - VIDEO REFRESH CONTROL***

**Purpose:** Specify whether the contents of video memory should be displayed on the screen; disabling refresh effectively blanks the screen.

**Registers at call:**
AH = 12h
BL = 36h
AL = new state (00h enabled, 01h disabled)

**Return Registers:**
AL = 12h if function supported

**Conflicts:** BIOS Window Extension (below)
**See also:** Compaq Function BF05h (below)

---

### INT 10h Function 13h

*AT and later, EGA, VGA*

### *WRITE STRING*

**Registers at call:**
AH = 13h
AL = write mode
  bit 0: update cursor after writing
  bit 1: string contains alternating characters and attributes
BH = page number
BL = attribute if string contains only characters
CX = number of characters in string
DH,DL = row,column at which to start writing
ES:BP -> string to write

**Return Registers:**
none

**Details:** Recognizes CR, LF, BS, and bell.

This function is also available on PC or XT with EGA or higher, since the video BIOS provides this function even on these older machines which do not support it in ROM BIOS.

The HP 95LX only supports write mode 00h.

**BUG:** On the IBM VGA Adapter, any scrolling which may occur is performed on the active page rather than the requested page.

**See also:** Functions 09h and 0Ah

---

### INT 10h Function 1400h

*Convertible, Compaq Portable 386*

### *LOAD USER-SPECIFIED LCD CHARACTER FONT*

**Registers at call:**
AX = 1400h
ES:DI -> character font
BH = number of bytes per character
  08h or 10h (Compaq)
BL = font to load
  00h main font (block 0)
  01h alternate font (block 1)
CX = number of characters to store
DX = character offset into RAM font area

**Return Registers:**
none

**See also:** Functions 11h and 1401h

---

### INT 10h Function 1401h

*Convertible, Compaq Portable 386*

### *LOAD SYSTEM ROM DEFAULT LCD CHARACTER FONT*

**Registers at call:**
AX = 1401h
BL = font to load
  00h main font (block 0)
  01h alternate font (block 1)

**Return Registers:**
none

**See also:** Functions 11h and 1400h

---

### INT 10h Function 1402h

*Convertible, Compaq Portable 386*

### *SET MAPPING OF LCD HIGH INTENSITY ATTRIBUTES*

**Registers at call:**
AX = 1402h
BL = function
  00h ignore high intensity attribute
  01h map high intensity to underscore
  02h map high intensity to reverse video
  03h map high intensity to selected alternate font

B0h half intensity (Compaq)
B1h toggle active intensity bit interpretation (Compaq Portable 386)

**Return Registers:**
none

---

### INT 10h Function 15h
### *GET PHYSICAL DISPLAY PARAMETERS*

*Convertible*

**Registers at call:**
AH = 15h

**Return Registers:**
AX = alternate display adapter type (see Table 8-6)
ES:DI -> display parameter table (see Table 8-7)

**Conflicts:** Sperry PC (below)
**See also:** Function 1Bh

**8-6 Values for alternate display adapter type:**

| | |
|---|---|
| 0000h | none |
| 5140h | LCD |
| 5153h | CGA |
| 5151h | mono |

**8-7 Format of display parameter table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | monitor model number |
| 02h | WORD | vertical pixels per meter |
| 04h | WORD | horizontal pixels per meter |
| 06h | WORD | total vertical pixels |
| 08h | WORD | total horizontal pixels |
| 0Ah | WORD | horizontal pixel separation in micrometers |
| 0Ch | WORD | vertical pixel separation in micrometers |

---

### INT 10h Function 1A00h
### *GET DISPLAY COMBINATION CODE*

*VGA, MCGA, PS/2*

**Registers at call:**
AX = 1A00h

**Return Registers:**
AL = 1Ah if function was supported
BL = active display code (see Table 8-8)
BH = alternate display code

**See also:** Function 12h Subfunction 35h, Function 1A01h, Function 1Bh

**8-8 Values for display combination code:**

| | |
|---|---|
| 00h | no display |
| 01h | monochrome adapter w/ monochrome display |
| 02h | CGA w/ color display |
| 03h | reserved |
| 04h | EGA w/ color display |
| 05h | EGA w/ monochrome display |
| 06h | PGA w/ color display |
| 07h | VGA w/ monochrome analog display |
| 08h | VGA w/ color analog display |
| 09h | reserved |
| 0Ah | MCGA w/ digital color display |
| 0Bh | MCGA w/ monochrome analog display |
| 0Ch | MCGA w/ color analog display |
| FFh | unknown display type |

---

### INT 10h Function 1A01h
*VGA, MCGA, PS/2*

### *SET DISPLAY COMBINATION CODE*

**Registers at call:**
AX = 1A01h
BL = active display code (see Table 8-8)
BH = alternate display code (see Table 8-8)

**Return Registers:**
AL = 1Ah if function was supported

**See also:** Function 12h Subfunction 35h, Function 1A00h

---

### INT 10h Function 1Bh
*VGA, MCGA, PS/2*

### *FUNCTIONALITY/STATE INFORMATION*

**Registers at call:**
AH = 1Bh
BX = implementation type
    0000h return funtionality/state information
ES:DI -> 64-byte buffer for state information (see Table 8-9)

**Return Registers:**
AL = 1Bh if function supported
    ES:DI buffer filled with state information

**BUG:** Trident 8900 (BIOS D3.0 11/12/91) and Trident 9000 (C3.0 10/25/91) do not correctly set the fields at offsets 27h and 29h of the state information

**See also:** Functions 15h, 1A00h, and 1F01h

**8-9 Format of state information:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | address of static funtionality table (see Table 8-12) |
| 04h | BYTE | video mode in effect |
| 05h | WORD | number of columns |
| 07h | WORD | length of regen buffer in bytes |
| 09h | WORD | starting address of regen buffer |
| 0Bh | WORD | cursor position for page 0 |
| 0Dh | WORD | cursor position for page 1 |
| 0Fh | WORD | cursor position for page 2 |
| 11h | WORD | cursor position for page 3 |
| 13h | WORD | cursor position for page 4 |
| 15h | WORD | cursor position for page 5 |
| 17h | WORD | cursor position for page 6 |
| 19h | WORD | cursor position for page 7 |
| 1Bh | WORD | cursor type |
| 1Dh | BYTE | active display page |
| 1Eh | WORD | CRTC port address |
| 20h | BYTE | current setting of register (3?8) |
| 21h | BYTE | current setting of register (3?9) |
| 22h | BYTE | number of rows |
| 23h | WORD | bytes/character |
| 25h | BYTE | display combination code of active display |
| 26h | BYTE | DCC of alternate display |
| 27h | WORD | number of colors supported in current mode |
| 29h | BYTE | number of pages supported in current mode |
| 2Ah | BYTE | number of scan lines active<br>(0,1,2,3) = (200,350,400,480)<br>Tseng ET3000: (4,5,6 = 512,600,768) |
| 2Bh | BYTE | primary character block |

| | | |
|---|---|---|
| 2Ch | BYTE | secondary character block |
| 2Dh | BYTE | miscellaneous flags (see Table 8-10) |
| 2Eh | 3 BYTEs | reserved (00h) |
| 31h | BYTE | video memory available<br>00h = 64K, 01h = 128K, 02h = 192K, 03h = 256K |
| 32h | BYTE | save pointer state flags (see Table 8-11) |
| 33h | 13 BYTEs | reserved (00h) |

**8-10 Bitfields for miscellaneous flags:**

| | |
|---|---|
| bit 0 | all modes on all displays on |
| bit 1 | gray summing on |
| bit 2 | monochrome display attached |
| bit 3 | default palette loading disabled |
| bit 4 | cursor emulation enabled |
| bit 5 | 0 = intensity; 1 = blinking |
| bit 6 | PS/2 P70 plasma display (without 9-dot wide font) active |
| bit 7 | reserved |

**8-11 Bitfields for save pointer state flags:**

| | |
|---|---|
| bit 0 | 512 character set active |
| bit 1 | dynamic save area present |
| bit 2 | alpha font override active |
| bit 3 | graphics font override active |
| bit 4 | palette override active |
| bit 5 | DCC override active |
| bits 6-7 | reserved |

**8-12 Format of Static Functionality Table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | modes supported #1<br>bit 0 to bit 7 = 1 modes 0,1,2,3,4,5,6 supported |
| 01h | BYTE | modes supported #2<br>bit 0 to bit 7 = 1 modes 8,9,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh supported |
| 02h | BYTE | modes supported #3<br>bit 0 to bit 3 = 1 modes 10h,11h,12h,13h supported<br>bit 4 to bit 7 reserved |
| 03h | BYTE | (IBM) reserved<br>(Tseng ET3000/4000) modes supported #4<br>bit 0 to bit 7 = modes 18h-1Fh supported |
| 04h | BYTE | (IBM) reserved<br>(Tseng ET3000/4000) modes supported #5<br>bit 0 to bit 7 = modes 20h-27h supported |
| 05h | BYTE | (IBM) reserved<br>(Tseng ET3000/4000) modes supported #6<br>bit 0 to bit 7 = modes 28h-2Fh supported |
| 06h | BYTE | (IBM) reserved<br>(Tseng ET3000/4000) modes supported #7<br>bit 0 to bit 7 = modes 30h-37h supported |
| 07h | BYTE | scan lines supported<br>bit 0 to bit 2 = 1 if scan lines 200,350,400 supported |
| 08h | BYTE | total number of character blocks available in text modes |
| 09h | BYTE | maximum number of active character blocks in text modes |

| | | |
|---|---|---|
| 0Ah | WORD | miscellaneous function support flags (see Table 8-13) |
| 0Ch | WORD | reserved |
| 0Eh | BYTE | save pointer function flags (see Table 8-14) |
| 0Fh | BYTE | reserved |

**8-13 Bitfields for miscellaneous function support flags:**

| | |
|---|---|
| bit 0 | all modes on all displays function supported |
| bit 1 | gray summing function supported |
| bit 2 | character font loading function supported |
| bit 3 | default palette loading enable/disable supported |
| bit 4 | cursor emulation function supported |
| bit 5 | EGA palette present |
| bit 6 | color palette present |
| bit 7 | color paging function supported |
| bit 8 | light pen supported |
| bit 9 | save/restore state function 1Ch supported |
| bit 10 | intensity blinking function supported |
| bit 11 | Display Combination Code supported |
| bits 12-15 | reserved |

**8-14 Bitfields for save pointer function flags:**

| | |
|---|---|
| bit 0 | 512 character set supported |
| bit 1 | dynamic save area supported |
| bit 2 | alpha font override supported |
| bit 3 | graphics font override supported |
| bit 4 | palette override supported |
| bit 5 | DCC extension supported |
| bits 6-7 | reserved |

## INT 10h Function 1Ch

*VGA, PS/2 Model 50+*

***SAVE/RESTORE VIDEO STATE***

| Registers at call: | Return Registers: |
|---|---|
| AH = 1Ch | AL = 1Ch if function supported |
| CX = requested states | |
| bit 0 video hardware | |
| bit 1 BIOS data areas | |
| bit 2 color registers and DAC state | |
| bits 3-15 reserved | |
| AL = function | |
| 00h return state buffer size | BX = number of 64-byte blocks needed |
| 01h save video state | |
| ES:BX -> buffer | |
| 02h restore video state | |
| ES:BX -> buffer containing previously saved state | |

**Details:** Many BIOSes corrupt the video registers when saving the state, so a program should restore the state immediately after saving it (the saved data is uncorrupted).

The BIOS data area consists of the 96 bytes from 0040h:0049h-00A8h.

**8-15 Format of video hardware state:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | sequencer index register |
| 01h | BYTE | CRTC index register |

| | | |
|---|---|---|
| 02h | BYTE | graphics controller index register |
| 03h | BYTE | attribute controller index register |
| 04h | BYTE | feature controller register |
| 05h | 4 BYTEs | sequencer registers |
| 09h | BYTE | sequencer register 0 |
| 0Ah | 25 BYTEs | CRTC registers 0-8 |
| 23h | 16 BYTEs | palette registers 00h-0Fh |
| 33h | 4 BYTEs | attribute registers 10h-13h |
| 37h | 9 BYTEs | graphics controller registers 0-8 |
| 40h | BYTE | CRTC base address (low) |
| 41h | BYTE | CRTC base address (high) |
| 42h | BYTE | plane 0 latch |
| 43h | BYTE | plane 1 latch |
| 44h | BYTE | plane 2 latch |
| 45h | BYTE | plane 3 latch |

**8-16 Format of DAC state:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | read/write mode DAC |
| 01h | BYTE | pixel address |
| 02h | BYTE | pixel mask |
| 03h | 768 BYTEs | color data (256 triples) |
| 303h | BYTE | color select register |

---

### INT 10h Function 1F00h

*XGA*

### *GET DMQS (Display Mode Query and Set) DATA LENGTH*

**Registers at call:**
AX = 1F00h

**Return Registers:**
AL = 1Fh if supported
BX = number of bytes of DMQS data

**Details:** This function is not supported on the original IBM XGA, only on XGA-NI (non-interlaced) and later models.

**See also:** Function 1F01h

---

### INT 10h Function 1F01h

*XGA*

### *READ DMQS DATA*

**Registers at call:**
AX = 1F01h
ES:DI -> user buffer for return data
(call Function 1F00h for size)

**Return Registers:**
AL = 1Fh if function supported
user buffer filled with DMQS data,
see Table 8-17.

**Details:** Not supported on the original IBM XGA, only on XGA-NI (non-interlaced) and later models.

**See also:** Functions 1Bh and 1F00h, 3270PC Video Function 3000h (NI)

**8-17 Format of DMQS buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | offset (in bytes) to DMQS data for next XGA instance |
| 02h | BYTE | slot number |
| 03h | BYTE | XGA implementation function level identifier |
| 04h | BYTE | XGA implementation resolution level identifier |
| 05h | WORD | vendor identifier - identifies card vendor |
| 07h | WORD | vendor defined field |
| 09h | WORD | XGA adapter I/O register base address |

| | | |
|---|---|---|
| 0Bh | WORD | XGA coprocessor register base address (paragraph--multiply by 10h to get physical address) |
| 0Dh | WORD | 1 Megabyte system video memory aperture (multiply by 100000h to get physical address)<br>0000h if not allocated |
| 0Fh | WORD | 4 Megabyte system video memory aperture (multiply by 100000h to get physical address)<br>0000h if not allocated |
| 11h | WORD | video memory base address (multiply by 100000h to get physical address) |
| 13h | WORD | composite ID of the attached display |
| 15h | BYTE | amount of video memory available, in multiples of 256K bytes |
| 16h | DWORD | alternate XGA coprocessor register base address (0 = none) |
| 1Ah | var | DMQS Data for further XGA Instances (as above) |

**Note:** " Instances " refers to the capability of having up to 8 XGA adapters in one computer.

## INT 1Dh

*SYSTEM DATA*

### *VIDEO PARAMETER TABLES*

**Details:** Default parameter table at F000h:F0A4h for 100% compatible BIOSes.

**See also:** INT 10h Function 00h

**8-18 Format of video parameters:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 16 BYTEs | 6845 register values for modes 00h and 01h |
| 10h | 16 BYTEs | 6845 register values for modes 02h and 03h |
| 20h | 16 BYTEs | 6845 register values for modes 04h and 05h |
| 30h | 16 BYTEs | 6845 register values for modes 06h and 07h |
| 40h | WORD | bytes in video buffer for modes 00h and 01h |
| 42h | WORD | bytes in video buffer for modes 02h and 03h |
| 44h | WORD | bytes in video buffer for modes 04h and 05h |
| 46h | WORD | bytes in video buffer for modes 06h and 07h |
| 48h | 8 BYTEs | columns on screen for each of modes 00h through 07h |
| 50h | 8 BYTEs | CRT controller mode bytes for each of modes 00h through 07h |

## INT 1Fh

*SYSTEM DATA*

### *8x8 GRAPHICS FONT*

**Purpose:** This vector points at 1024 bytes of graphics data, 8 bytes for each character 80h-FFh.

**Details:** The graphics data for characters 00h-7Fh is stored at F000h:FA6Eh in 100% compatible BIOSes.

**See also:** INT 43h, AX PC INT 10h Function 5000h (Chapter 6)

## INT 42h

*EGA, VGA*

### *RELOCATED DEFAULT INT 10h VIDEO SERVICES*

**Purpose:** This vector points at the ROM BIOS INT 10h handler which was in effect when the EGA/VGA BIOS initialized.

**Conflicts:** Acorn BBS Master (Chapter 7), Z100 (Chapter 7), Western Digital SuperBIOS (Chapter 10)

**See also:** INT 10h

**Details:** This vector is not used by the PS/2 built-in VGA or XGA.

## INT 43h

*VIDEO DATA*

### *CHARACTER TABLE*

*EGA, MCGA, VGA*

**Purpose:** This vector points at graphics data for characters 00h-7Fh of the current font.

**Conflicts:** Acorn BBS Master (Chapter 7), Z100 (Chapter 7)
**See also:** INT 1Fh, INT 44h

---

**INT 44h** *VIDEO DATA*
*PCjr*

### *ROM BIOS CHARACTER FONT, CHARACTERS 00h-7Fh*

**Purpose:** This vector points at graphics data for the current character font.
**Conflicts:** Virus Acorn BBS Master (Chapter 7), Z100 (Chapter 7), Virus (Chapter 59), IBM 3270-PC (NI), Novell NetWare (NI)
**See also:** INT 1Fh, INT 43h

---

**INT 6Dh** *VGA*
***internal***

**Details:** Used by IBM, Paradise, Video7, and NCR.
**Conflicts:** Trident (Chapter 9), ATI VGA Wonder (Chapter 9)

---

**INT 7Fh** *IBM XGA*

### *UNDETERMINED FUNCTION*

**Conflicts:** Halo88 (below), HDILOAD (below), Touch Screen (below), TIGA (Chapter 12), GO32 (Chapter 37), Multilink Advanced (Chapter 49), Btrieve (Chapter 66), Canon IXHND2 (Chapter 66), YTERM (Chapter 66), Alloy 386/Multiware (NI), Convergent CTOS (NI), HLLAPI (NI), Telebit ACS (NI), Novell NetWare (NI)

## Vendor-Specific Extensions

**INT 10h Function 0070h** *Everex Micro Enhancer EGA/Viewpoint VGA*

### *EXTENDED MODE SET*

**Purpose:** Specify a proprietary display mode on the Everex Micro Enhancer or Viewpoint video adapters.

**Registers at call:**
AX = 0070h
BL = mode (see Table 8-19)

**Return Registers:**
none

**Conflicts:** Standard Function 00h (above)
**See also:** Function 00h, Video7 Function 6F05h (below), Everex Function 7000h Subfunction 0004h (below), GO32 Function FFh (Chapter 37)

**8-19 Values for video mode:**

| | *text/ grph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disp page* | *scrn addr* | *monitor* | *adapter* |
|---|---|---|---|---|---|---|---|---|---|
| 00h | = G | | | 640x480 | 16 | | | multsync | EGA,VGA |
| 01h | = G | | | 752x410 | 16 | | | multsync | EGA,VGA |
| 02h | = G | | | 800x600 | 16 | | | multsync | EGA,VGA |
| 03h | = T | 80x34 | | | | | | multsync | EGA,VGA |
| 04h | = T | 80x60 | | | | | | multsync | EGA,VGA |
| 05h | = T | 94x29 | | | | | | multsync | EGA only |
| 06h | = T | 94x51 | | | | | | multsync | EGA only |
| 07h | = T | 100x43 | 8x14 | | 16 | | | | VGA only |
| 08h | = T | 100x75 | 8x8 | | 16 | | | | VGA only |
| 09h | = T | 80x44 | | | | | | EGA | EGA only |
| 0Ah | = T | 132x25 | | | | | | EGA | EGA,VGA |
| 0Bh | = T | 132x44 | | | | | | EGA | EGA,VGA |
| 0Ch | = T | 132x25 | | | | | | CGA | EGA only |
| 0Dh | = T | 80x44 | | | mono | | | | EGA only |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0Eh | = T | 132x25 | | | mono | |
| 0Fh | = T | 132x44 | | | mono | |
| 10h | = | reserved | | | | |
| 11h | = G | | | 1280x350 | 4 | EGA only |
| 12h | = G | | | 1280x600 | 4 | EGA only |
| 13h | = G | | | 640x350 | 256 | EGA, EV673 |
| 14h | = G | | | 640x400 | 256 | |
| 15h | = G | | | 512x480 | 256 | |
| 16h | = T | 80x30 | 8x16 | | 256 | VGA only |
| 18h | = T | 100x27 | 8x16 | | 16 | VGA only |
| 20h | = G | | | 1024x768 | 16 | Everex 629,678<br>Everex EV-673 |
| 21h | = T | 160x64 | 8x16 | 1280x1024 | 16 | 1MB VGA only |
| 30h | = G | | | 640x480 | 256 | Everex 629,678<br>Everex EV-673 |
| 31h | = G | | | 800x600 | 256 | Everex 629,678<br>Everex EV-673 |
| 32h | = G | 128x48 | 8x16 | 1024x768 | 256 | 1MB VGA only |
| 40h | = T | 132x30 | 8x16 | | 16 | VGA only |
| 50h | = T | 132x32 | 8x16 | | mono | VGA only |
| 62h | = G | 40x25 | 8x8 | 320x200 | 32K | Vwpt TC (EV629) |
| 70h | = G | 64x30 | 8x16 | 512x480 | 32K | Viewpoint TC |
| 71h | = G | 80x30 | 8x16 | 640x480 | 32K | Viewpoint TC |
| 76h | = G | 64x30 | 8x16 | 512x480 | 16M | Viewpoint TC |
| 77h | = G | 80x30 | 8x16 | 640x480 | 16M | Viewpoint TC |

**See also:** Tables 8-2, 8-27, 8-37, 9-4, and 9-5

---

### INT 10h Function 007Eh

*Paradise VGA, AT&T VDC600*

***SET SPECIAL MODE***

**Purpose:** Specify a display mode by resolution rather than mode number.

**Registers at call:**

AX = 007Eh

BX = horizontal dimension of the mode desired

CX = vertical dimension of the mode desired

DX = number of colors of the mode desired (0000h for monochrome modes)

**Return Registers:**

BH = 7Eh if successful (Paradise VGA)

AL = 7Eh if successful (AT&T VDC600)

**Details:** Both BX and CX are in pixels for graphics modes, rows for text modes.

**Conflicts:** Standard Function 00h (above)

**See also:** Functions 0070h and 007Fh, Standard Function 00h (above), Video7 Function 6F05h (below), GO32 Function FFh (Chapter 37)

---

### INT 10h Function 007Fh

*Paradise VGA, AT&T VDC600*

***EXTENDED FUNCTIONS***

**Purpose:** Set the emulation mode or access proprietary extended data registers.

**Registers at call:**

AX = 007Fh

BH = function

00h set VGA operation

01h set non-VGA operation
color modes (0,1,2,3,4,5,6) will set non-VGA CGA operation
monochrome mode 7 will set non-VGA MDA/Hercules operation
02h query mode status
03h lock current mode
allows current mode (VGA or non-VGA) to survive re-boot
04h enter CGA mode (AT&T VDC600 only)
05h enter MDA mode (AT&T VDC600 only)
0Ah,0Bh,0Ch,0Dh,0Eh,0Fh WRITE PARADISE REGISTERS 0,1,2,3,4,5
(port 03CEh indices 0Ah,0Bh,0Ch,0Dh,0Eh,0Fh)
BL = value to set in the paradise register
1Ah,1Bh,1Ch,1Dh,1Eh,1Fh READ PARADISE REGISTERS 0,1,2,3,4,5
(port 03CEh indices 0Ah,0Bh,0Ch,0Dh,0Eh,0Fh)

**Return Registers:**
BH = 7Fh if successful
AL = 7Fh if successful (AT&T VDC600)
BL = 00h if operating in VGA mode, 01h if non-VGA mode
CH = total video RAM size in 64k byte units
CL = video RAM used by the current mode
BL = value of the paradise register

**Conflicts:** Standard Function 00h (above)
**See also:** Function 007Eh, Function FFh

---

## INT 10h Function 10h
### *SET SCROLL SPEED*

*Eagle PC2 BIOS Rev. C*

**Registers at call:**
AH = 10h
AL = speed
00h fast
01h slow (scrolling only moves characters during vertical retrace)

**Return Registers:**
AH = previous speed

**Conflicts:** standard Functions 10xxh (above), BIOS Window Extension (below)

---

## INT 10h Function 11h
### *TEXT-MODE CHARACTER GENERATOR FUNCTIONS*

*Realtek RTVGA*

**Registers at call:**
AH = 11h
AL = 07h load user-specified paterns and recalculate mode parms
parameters are the same as for standard Function 1110h
AL = 08h load monochrome patterns (8x14) and recalculate mode parms
parameters are the same as for standard Function 1111h
AL = 09h load ROM 8 by 8 double-dot patterns and recalculate mode parms
parameters are the same as for standard Function 1112h
AL = 0Bh load ROM 8x16 character set (VGA) and recalculate mode parms
parameters are the same as for standard Function 1114h

**Return Registers:**
none

**Details:** These functions should only be called under the same conditions as AL=1xh (see entry above).
**Conflicts:** BIOS Window Extension (below)
**See also:** Standard Functions 11xxh (above)

---

## INT 10h Function 110Fh
Realtek RTVGA
### SET USER 8x8 GRAPHICS CHARACTERS

**Registers at call:**
AX = 110Fh
ES:BP -> user table for INT 1Fh

**Return Registers:**
none

**Details:** This function is meant to be called immediately after a mode set; results are unpredictable at other times.
**Conflicts:** BIOS Window Extension (below)
**See also:** Standard Functions 1110h through 1112h (above), standard Function 1120h (above)

---

## INT 10h Function 1110h
Realtek RTVGA
### SET USER GRAPHICS CHARACTERS

**Registers at call:**
AX = 1110h
ES:BP -> user table
CX = bytes per character
BL = row specifier
  00h user set
    DL = number of rows
  01h 14 rows
  02h 25 rows
  03h 43 rows

**Return Registers:**
none

**Details:** This function is meant to be called immediately after a mode set; results are unpredictable at other times.
**Conflicts:** Standard Function 1110h (above), BIOS Window Extension (below)
**See also:** Functions 110Fh, 1111h, and 1112h; standard Function 1121h (above)

---

## INT 10h Function 1111h
Realtek RTVGA
### SET ROM 8x14 GRAPHICS CHARACTERS

**Registers at call:**
AX = 1111h
BL = row specifier (see Function 1110h)

**Return Registers:**
none

**Details:** This function is meant to be called immediately after a mode set; results are unpredictable at other times.
**Conflicts:** Standard Function 1111h (above), BIOS Window Extension (below)
**See also:** Functions 110Fh, 1110h, and 1122h

---

## INT 10h Function 1112h
Realtek RTVGA
### SET ROM 8x8 DOUBLE-DOT CHARACTERS

**Registers at call:**
AX = 1112h
BL = row specifier (see Function 1110h)

**Return Registers:**
none

**Details:** This function is meant to be called immediately after a mode set; results are unpredictable at other times.
**Conflicts:** Standard Function 1112h (above), BIOS Window Extension (below)
**See also:** Functions 110Fh through 1111h, standard Function 1123h (above)

### INT 10h Function 1129h

*Compaq Systempro*

### *GRAPH-MODE CHARGEN - LOAD 8x16 GRAPHICS CHARACTERS*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1129h | none |
| BL = row specifier (see Function 1110h) | |

**Details:** This function is meant to be called immediately after a mode set; results are unpredictable at other times.

UltraVision v2+ sets INT 43h to the appropriate font on this call.

**Conflicts:** BIOS Window Extension (below)

**See also:** standard Functions 1120h through 1124h (above), INT 1Fh (above), INT 43h (above)

### INT 10h Function 12h, Subfunction 55h

*ATI, Tatung, Taxan*

### *ALTERNATE FUNCTION SELECT - ENHANCED FEATURES*

| Registers at call: | Return Registers: |
|---|---|
| AH = 12h | |
| BH = 55h | |
| BL = subfunction | |
| 00h disable enhanced features | |
| 01h enable enhanced features | |
| 02h get status | AL = status flags (see Table 8-20) |
| 03h disable register trapping (CGA emulation) | |
| 04h enable register trapping | |
| 05h program video mode | |
| ES:BP -> video mode table (see Table 8-21) | ES:BP -> table suitable for mode AL (and subfunction BL=05h) |
| 06h get mode table | BP = FFFFh on error |
| AL = video mode | |

**Conflicts:** BIOS Window Extension (below)

**See also:** Function FFh

**8-20 Bitfields for status flags:**

| | |
|---|---|
| bit 3 | set if enhanced features enabled |
| bits 7-5 | monitor type<br>000 PS/2 mono<br>001 PS/2 color<br>010 multi-sync<br>011 Taxan 650 25kHz<br>100 RGB<br>101 mono<br>110 EGA<br>111 Compaq internal |

**8-21 Format of ATI VGA Wonder video mode table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | number of columns |
| 01h | BYTE | maximum row (number of rows - 1) |
| 02h | BYTE | scan lines per row |
| 03h | WORD | video buffer size in bytes |
| 05h | 4 BYTEs | values for Sequencer registers 1-4 |
| 09h | BYTE | value for Miscellaneous Output register |
| 0Ah | 25 BYTEs | values for CRTC registers 00h-18h |

| | | |
|---|---|---|
| | | 00h horizontal total size (chars)<br>01h horizontal displayed (chars)<br>02h horizontal sync position (chars)<br>03h horizontal sync width (chars)<br>04h vertical total size (char rows)<br>05h vertical total adjust (scan lines)<br>06h vertical displayed (char rows)<br>07h vertical sync position (char rows)<br>08h interlace mode<br>09h max scan line in row<br>0Ah cursor start scan line<br>0Bh cursor end scan line<br>0Ch screen memory start (high)<br>0Dh screen memory start (low)<br>0Eh cursor address (high)<br>0Fh cursor address (low)<br>10h light pen (high)<br>11h light pen (low) |
| 23h | 20 BYTEs | default palette (values for Attribute Controller regs 00h-13h) |
| 37h | 9 BYTEs | values for Graphics Controller registers 00h-08h |

---

### INT 10h Function 15h *Sperry PC*
### *SET SUPERIMPOSE MODE*

**Registers at call:**
AH = 15h
AL = superimpose mode
  00h show graphics screen
  01h show text screen
  02h show text screen superimposed on graphics screen

**Return Registers:**
none

**Conflicts:** IBM Convertible (above)

---

### INT 10h Function 5F00h *Chips & Technologies Extended BIOS*
### *RETURN CHIP INFORMATION*

**Registers at call:**
AX = 5F00h

**Return Registers:**
AL = 5Fh function supported
  BL = chip type (see Table 8-22)
  BH = video memory size (00h 256KB, 01h 512KB, 02h 1MB)
  CX = miscellaneous flags
    bits 15-2 reserved
    bit 1 = system type (0=PC/AT, 1=PS/2)
    bit 2 = DAC size (0=6-bit, 1=8-bit)

**Conflicts:** Realtek RTVGA below
**See also:** C&T Function 5F01h

**8-22 Bitfields for chip type:**

| | |
|---|---|
| bits 7-4 | chip type<br>0000: 82c451<br>0001: 82c452<br>0010: 82c455 |
| bits 3-0 | revision number |

---

### INT 10h Function 5F00h

*Realtek RTVGA*

### *RETURN CHIP VERSION*

**Registers at call:**
AX = 5F00h

**Return Registers:**
AH = 00h, if successful
AL = chip version (the same value that VTEST.EXE reads)
    00h *RTG3103*
    01h RTG31030/RTG3105
    02h *RTG3106*
    3Fh non-Realtek chip

**BUG:** In v3.C10, AX=5F00h on return due to improper stack restoration code.
**Conflicts:** C&T above
**See also:** RTVGA Function 5F01h

---

### INT 10h Function 5F01h

*Chips & Technologies Extended BIOS*

### *SET EMULATION*

**Registers at call:**
AX = 5F01h
BL = emulation type (see Table 8-23)

**Return Registers:**
AL = 5Fh if function supported
    AH = status
        00h unsuccessful
        01h successful

**Conflicts:** Realtek RTVGA below
**See also:** C&T Functions 5F00h and 5F02h, Oak VGA Function FFh

**8-23 Values for emulation type:**

| | |
|---|---|
| 00h,01h | reserved |
| 02h | CGA |
| 03h | MDA |
| 04h | Hercules |
| 05h | EGA |
| 06h | VGA |

---

### INT 10h Function 5F01h

*Realtek RTVGA*

### *WRITE RTVGA BIOS STRING TO DESTINATION*

**Registers at call:**
AX = 5F01h
ES:DI -> zero-filled buffer for BIOS ID string

**Return Registers:**
AH = 00h if succesful
ES:DI -> ASCII signature " REALTEK VGA BIOS Version 3C.10 "

**BUG:** In v3.C10, AX=5F01h on return due to improper stack restoration code.
**Conflicts:** C&T above
**See also:** RTVGA Functions 5F00h and 5F02h

---

### INT 10h Function 5F02h

*Chips & Technologies Extended BIOS*

### *AUTO EMULATION*

**Registers at call:**
AX = 5F02h
BL = new state of autmatic emulation (00h enabled, 01h disabled)

**Return Registers:**
AL = 5Fh if function supported
    AH = status
        00h unsuccessful
        01h successful

**Conflicts:** Realtek RTVGA below
**See also:** C&T Functions 5F01h and 5F03h

---

### INT 10h Function 5F02h

*Realtek RTVGA*

**RETURN RTVGA ON-BOARD MEMORY SIZE**

**Registers at call:**
AX = 5F02h

**Return Registers:**
AH = 00h, if succesful
AL = on-board memory size
(00h = 256K, 01h = 512K, 02h = 768K, 03h = 1024K)

**BUG:** In v3.C10, AX=5F02h on return due to improper stack restoration code.
**Conflicts:** C&T above
**See also:** RTVGA Functions 5F01h and 5F03h

---

### INT 10h Function 5F03h

*Realtek RTVGA (undocumented)*

**SET UNKNOWN VALUE**

**Registers at call:**
AX = 5F03h
BL = 0-3 (*unknown value*)

**Return Registers:**
AH = 00h if succesful

**BUG:** In v3.C10, AX=5F03h on return due to improper stack restoration code.
**See also:** RTVGA Function 5F02h

---

### INT 10h Function 5F03h, Subfunction 00h

*Chips & Technologies Extended BIOS*

**SET POWER-ON DISPLAY MODE**

**Registers at call:**
AX = 5F03h
BL = 00h
CL = display mode
CH = mode
    bits 1-0 scanlines
        00=200,01=350,10=400
    bit 7 persistence (0 reset on next boot, 1 until changed)

**Return Registers:**
AL = 5Fh if function supported
    AH = status
        00h unsuccessful
        01h successful

**See also:** C&T Functions 5F02h and 5F90h, C&T Function 5F03h Subfunction 01h

---

### INT 10h Function 5F03h, Subfunction 01h

*Chips & Technologies Extended BIOS*

**SET POWER-ON EMULATION MODE**

**Registers at call:**
AX = 5F03h
BL = 01h
CL = emulation mode
CH = mode
    bit 7 persistence (0 reset on next boot, 1 until changed)

**Return Registers:**
AL = 5Fh if function supported
    AH = status
        00h unsuccessful
        01h successful

**See also:** C&T Functions 5F02h and 5F90h, C&T Function 5F03h Subfunction 00h

---

### INT 10h Functions 5F90h to 5F92h

*Chips & Technologies Extended BIOS*

**SAVE/RESTORE SUPERVGA STATE**

**Registers at call:**
AH = 5Fh
AL = subfunction
    90h determine size of save buffer
    91h save state
    92h restore state
CX = state mask (see Table 8-24)

ES:BX -> save buffer (subfunctions 91h,92h only)

**Return Registers:**
AL = 5Fh if function supported
**subfunction 90h**
BX = number of 64-byte blocks required
**subfunction 92h**
buffer at ES:BX filled

**See also:** C&T Function 5F03h, standard Function 1Ch (above)

**8-24 Bitfields for state mask:**

| | |
|---|---|
| bit 0 | video hardware |
| bit 1 | BIOS data state |
| bit 2 | DAC state |
| bit 15 | type (0 all state info, 1 SuperVGA state only) |

---

### INT 10h Function 6E00h

*Paradise VGA internal*

#### *GET UNKNOWN INFORMATION*

**Registers at call:**
AX = 6E00h

**Return Registers:**
AH = *unknown* (depends on serial number)
AL = *unknown*
BX = *unknown*
CL = *unknown*
CH = *unknown*
DX = *unknown*

**See also:** Functions 6E04h and 6E05h

---

### INT 10h Function 6E04h

*Paradise VGA internal*

#### *GET SCREEN SIZE AND UNKNOWN VALUES*

**Registers at call:**
AX = 6E04h

**Return Registers:**
BX = screen width (columns)
CX = screen height (lines)
AH = *unknown* (05h or FFh)
AL = *unknown* (04h or video mode)

**See also:** Functions 6E00h and 6E05h

---

### INT 10h Function 6E05h

*Paradise VGA internal*

#### *SET MODE*

**Registers at call:**
AX = 6E05h
BL = mode

**Return Registers:**
none

**Details:** Like Function 00h with AL=BL.

**See also:** Functions 6E00h, 6E04h, and 6F05h; Standard Function 00h

---

### INT 10h Function 6F00h

*Video7,VEGA,HP Ext BIOS*

#### *INSTALLATION CHECK*

**Purpose:** Determine whether the Video7 VGA, VEGA VGA, or HP Extended BIOS video functions are present.

**Registers at call:**
AX = 6F00h
BX = 0000h

**Return Registers:**
BX = 5637h ('V7') indicates Video7 VGA/VEGA VGA extensions are present
BX = 4850h ('HP') indicates HP Extended BIOS video functions present

**See also:** Functions 6F01h to 6F07h

### INT 10h Function 6F01h
*Video7 VGA, VEGA VGA, HP Ext. BIOS*

### *GET MONITOR INFO*

**Registers at call:**
AX = 6F01h

**Return Registers:**
AL = monitor type code (HP,VEGA VGA only)
(see Table 8-25)
AH = status register information
bit 0 = display enable
0 display enabled
1 vertical or horizontal retrace in progress
bit 1 = light pen flip flop set
bit 2 = light pen switch activated
bit 3 = vertical sync
bit 4 = monitor resolution
0 high resolution (over 200 lines)
1 low resolution (up to 200 lines)
bit 5 = display type
0 color
1 monochrome
bits 6,7 = diagnostic bits
CL = current value of Extended Control register (HP Extended BIOS, and only if AL=41h)

**Details:** Bits 0-3 are the same as the EGA/VGA status register bits 0-3.

**8-25 Values for monitor type code (HP Extended BIOS):**

| | |
|---|---|
| 00h | non-HP card with ROM and possibly its own INT 10h driver |
| 41h | MultiMode video display adapter |
| 42h-44h | reserved |
| 45h | industry standard monochrome display adapter |
| 46h | industry standard color display adapter |
| 51h | reserved |

### INT 10h Function 6F02h
*HP Vectra Extended BIOS*

### *SET MONITOR INFO*

**Registers at call:**
AX = 6F02h
BL = new value for extended control register
(see Table 8-26)

**Return Registers:**
none

**Details:** This function is only valid when an HP MultiMode Video Display Adapter is installed. The Extended Control register is at I/O address 3DDh.

**See also:** Functions 6F00h, 6F01h, and 6F03h

**8-26 Bitfields for extended control register:**

| | |
|---|---|
| bit 0 | screen resolution (0 = 200 lines, 1 = 400 lines) |
| bit 1 | underline enable (if set, 'blue' bit of fg color = underl) |
| bit 2 | font (0 = Standard-8, 1 = HP-Roman-8) |
| bit 3 | memory disabled for CPU access |
| bit 4 | allow access to full 32K memory instead of wrapping at 16K |
| bit 5 | select second 16K page instead of first |
| bits 6,7 | unused |

### INT 10h Function 6F03h

*HP Vectra Extended BIOS*

### *MODIFY MONITOR INFO*

**Registers at call:**
AX = 6F03h
BH = exclude mask (set bits are not modified)
BL = new values for bits indicated by BH (see Table 8-26)

**Return Registers:**
none

**Details:** This function is only valid when an HP MultiMode Video Display Adapter is installed.

**See also:** Functions 6F00h, 6F01h, and 6F02h

### INT 10h Function 6F04h

*Video7 VGA, VEGA VGA, HP Vectra*

### *GET MODE AND SCREEN RESOLUTION*

**Registers at call:**
AX = 6F04h

**Return Registers:**
AL = current video mode (see Table 8-27)
BX = horizontal columns (text) or pixels (graphics)
CX = vertical columns (text) or pixels (graphics)

**See also:** Functions 6F00h and 6F05h

### INT 10h Function 6F05h

*Video7 VGA, VEGA EGA/VGA, HP Vectra*

### *SET VIDEO MODE*

**Registers at call:**
AX = 6F05h
BL = new video mode (see Table 8-27)

**Return Registers:**
none

**Details:** On the HP Vectra, this function rather than Function 00h must be used to return to an IBM-standard mode after setting an HP-specified mode from 08h to 0Fh.

**See also:** Functions 0070h, 007Eh, and 6F04h; standard Function 00h (above)

**8-27 Values for video mode:**

| | *text/ graph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disp page* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| 00h-13h | = standard IBM modes (see standard Function 00h) | | | | | | | |
| 08h | = T | 80x27 | | | mono | | | HP MultiMode Video |
| 09h | = T | 80x27 | | | | | | HP MultiMode Video |
| 0Ah | = T | 40x27 | | | mono | | | HP MultiMode Video |
| 0Bh | = T | 40x27 | | | | | | HP MultiMode Video |
| *0Ch* | = | *Reserved* | | | | | | *HP MultiMode Video* |
| 0Dh | = G | | | 640x400 | | | | HP MultiMode Video |
| 0Eh | = G | | | 320x400 | | | | HP MultiMode Video |
| 0Fh | = G | | | 320x400 | | | | HP MultiMode Video |
| 40h | = T | 80x43 | 8x8 | | | | | Video7/VEGA VGA |
| 41h | = T | 132x25 | 8x14 | | | | | Video7/VEGA VGA |
| 42h | = T | 132x43 | 8x8 | | | | | Video7/VEGA VGA |
| 43h | = T | 80x60 | 8x8 | | | | | Video7/VEGA VGA |
| 44h | = T | 100x60 | 8x8 | | | | | Video7/VEGA VGA |
| 45h | = T | 132x28 | 8x8 | | | | | Video7/VEGA VGA |
| 60h | = G | | | 752x410 | 16 | | | Video7 VGA, VEGA VGA |
| 61h | = G | | | 720x540 | 16 | | | Video7 VGA, VEGA VGA |
| | = G | | | 720x540 | 16 | | | Northgate, Headland 1024i |

| | *text/ graph* | *text resol* | *pixel box* | *pixel resoltn* | *colors* | *disp page* | *scrn addr* | *system* |
|---|---|---|---|---|---|---|---|---|
| 62h | = G | | | 800x600 | 16 | | | Video7 VGA, VEGA Ext EGA |
| | = G | | | 800x600 | 16 | | | Headland 1024i |
| 63h | = G | | | 1024x768 | 2 | | | Video7 VGA |
| 64h | = G | | | 1024x768 | 4 | | | Video7 VGA |
| 65h | = G | | | 1024x768 | 16 | | | Video7 VGA, VEGA Ext EGA |
| | = G | | | 1024x768 | 16 | | | Headland 1024i |
| 66h | = G | | | 640x400 | 256 | | | Video7 VGA, VEGA Ext VGA |
| | = G | | | 640x400 | 256 | | | Northgate, Headland 1024i |
| 67h | = G | | | 640x480 | 256 | | | Video7 VGA, VEGA Ext VGA |
| | = G | | | 640x480 | 256 | | | Headland 1024i |
| 68h | = G | | | 720x540 | 256 | | | Video7 VGA, VEGA Ext VGA |
| | = G | | | 720x540 | 256 | | | Headland 1024i |
| 69h | = G | | | 800x600 | 256 | | | Video7 VGA, VEGA Ext VGA |
| | = G | | | 800x600 | 256 | | | Headland 1024i |
| 70h | = G | | | 752x410 | 16 gray | | | Video7 VGA, VEGA VGA |
| 71h | = G | | | 720x540 | 16 gray | | | Video7 VGA, VEGA VGA |
| 72h | = G | | | 800x600 | 16 gray | | | Video7 VGA |
| 73h | = G | | | 1024x768 | 2 gray | | | Video7 VGA |
| 74h | = G | | | 1024x768 | 4 gray | | | Video7 VGA |
| 75h | = G | | | 1024x768 | 16 gray | | | Video7 VGA |
| 76h | = G | | | 640x400 | 256 gray | | | Video7 VGA |
| 77h | = G | | | 640x480 | 256 gray | | | Video7 VGA |
| *78h* | *= G* | | | *720x540* | *256 gray* | | | *Video7 VGA* |
| 79h | = G | | | 800x600 | 256 gray | | | (future) |

**See also:** Tables 8-2, 8-19, 8-37, 9-4, and 9-5

---

## INT 10h Function 6F06h

*Video7 VGA, VEGA VGA*

### *SELECT AUTOSWITCH MODE*

**Registers at call:**
AX = 6F06h
BL = Autoswitch mode select
  00h select EGA/VGA-only modes
  01h select Autoswitched VGA/EGA/CGA/MGA modes
  02h select 'bootup' CGA/MGA modes
BH = enable/disable (00h enable, 01h = disable selection)

**Return Registers:**
none

### INT 10h Function 6F07h

*Video7 VGA, VEGA VGA*

### *GET VIDEO MEMORY CONFIGURATION*

**Registers at call:**
AX = 6F07h

**Return Registers:**
AL = 6Fh if supported
AH = memory information
  bits 0-6 = number of 256K blocks of video memory
  bit 7 = DRAM/VRAM (0: DRAM, 1: VRAM)
BH = chip revision (SR8F) (S/C Chip in VEGA VGA)
BL = chip revision (SR8E) (G/A Chip in VEGA VGA)
CX = 0000h

**See also:** Standard Function 12h Subfunction 10h (above)

---

### INT 10h Function 70h

*Tandy 2000 only*

### *GET ADDRESS OF VIDEO RAM*

**Registers at call:**
AH = 70h

**Return Registers:**
AX:BX -> WORD containing green plane's offset
AX:CX -> WORD containing green plane's segment
AX:DX -> WORD containing segment of red (offset 0) and blue (offset 4000) planes

**Conflicts:** Everex Functions 7000h below
**See also:** Tandy Function 71h

---

### INT 10h Function 7000h, Subfunction 0000h

*Everex Extended Video BIOS*

### *RETURN EMULATION STATUS*

**Registers at call:**
AX = 7000h
BX = 0000h

**Return Registers:**
AL = 70h if Trident-based Everex card
CL = monitor type (see Table 8-28)
CH = feature bits (see Table 8-29)
DX = video board info
    bits 4-15: board ID model
    bits 0-3: board ID revision
DI = BCD BIOS version number

**Details:** Board models for Trident-based Everex cards: 236h = Ultragraphics II, 620h = Vision VGA, 673h = EVGA, 678h = Viewpoint.
**Conflicts:** Tandy Function 70h
**See also:** C&T Function 5F01h, Oak VGA Function FFh

**8-28 Values for monitor type:**

| | |
|---|---|
| 00h | mono |
| 01h | CGA |
| 02h | EGA |
| 03h | digital multifrequency |
| 04h | IBM PS/2 |
| 05h | IBM 8514 |
| 06h | SuperVGA |
| 07h | analog multifrequency |
| 08h | super multifrequency |

**8-29 Bitfields for feature bits:**

| | |
|---|---|
| bits 7,6 | memory size<br>00 = 256K<br>01 = 512K<br>10 = 1024K |

| | |
|---|---|
| | 11 = 2048K |
| bit 5 | special oscillator present |
| bit 4 | VGA protect enabled |
| bit 0 | 6845 emulation |

---

### INT 10h Function 7000h, Subfunction 0004h
*Everex Extended Video BIOS*

### *GET PAGING FUNCTION POINTER FOR CURRENT MODE*

**Registers at call:**
AX = 7000h
BX = 0004h

**Return Registers:**
ES:DI -> FAR paging function (call with DL = page to set)

**Details:** The word preceding ES:DI is the length of the function in bytes, and the last byte of the function is a FAR return instruction.

**Conflicts:** Tandy Function 70h

**See also:** Function 7000h Subfunctions 0000h and 0005h, VESA Function 4F05h (Chapter 9)

---

### INT 10h Function 7000h, Subfunction 0005h
*Everex Extended Video BIOS*

### *GET SUPPORTED MODE INFO*

**Registers at call:**
AX = 7000h
BX = 0005h
CL = maximum number of modes to get info for
CH = mode type to get info for (see Table 8-30)
DL = monitor type to get info for
ES:DI -> buffer for mode info record (see Table 8-31)

**Return Registers:**
CL = total number of modes fitting criteria
CH = size of each info record

**Conflicts:** Tandy Function 70h

**See also:** Function 7000h Subfunctions 0000h and 0004h

**8-30 Values for mode type:**

| | |
|---|---|
| 00h | all modes |
| 01h | monochrome text modes |
| 02h | color text modes |
| 03h | four-color CGA graphics modes |
| 04h | two-color CGA graphics modes |
| 05h | 16-color graphics modes |
| 06h | 256-color graphics modes |

**8-31 Format of mode information record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | mode number (bit 7 set if extended mode) |
| 01h | BYTE | mode type (see Table 8-30) |
| 02h | BYTE | info bits |
| | | bits 7,6 reserved |
| | | bit 5 monochrome mode |
| | | bit 4 interlaced display |
| | | bit 3 requires special oscillator |
| | | bits 2,1 memory required |
| | | 00 = 256K |
| | | 01 = 512K |
| | | 10 = 1024K |
| | | 11 = 2048K |
| | | bit 0 reserved |
| 03h | BYTE | font height |

| | | |
|---|---|---|
| 04h | BYTE | text columns on screen |
| 05h | BYTE | text rows on screen |
| 06h | WORD | number of scan lines |
| 08h | BYTE | color information<br>bits 7-4 reserved<br>bits 3-0 bits per pixel |

---

### INT 10h Function 71h
### *GET ADDRESS OF INCRAM*

*Tandy 2000 only*

**Registers at call:**
AH = 71h

**Return Registers:**
AX:BX -> WORD containing segment address of INCRAM
AX:CX -> WORD containing offset of INCRAM

**See also:** Tandy Function 70h

---

### INT 10h Function 72h
### *SCROLL RIGHT PART OR ALL OF SCREEN*

*Tandy 2000 only*

**Registers at call:**
AH = 72h
AL = number of columns to shift scroll area, 00h to clear entire area
BH = new attributes for blanked columns at left
CH,CL = row, column of upper left corner of scroll area
DH,DL = row, column of lower right corner of scroll area

**Return Registers:**
none

**See also:** Tandy Function 73h, standard Functions 06h and 07h, TopView INT 15h Function 12h Subfunction 05h (Chapter 44)

---

### INT 10h Function 73h
### *SCROLL LEFT PART OR ALL OF SCREEN*

*Tandy 2000 only*

**Registers at call:**
AH = 73h
AL = number of columns to shift scroll area, 00h to clear entire area
BH = new attributes for blanked columns at right
CH,CL = row, column of upper left corner of scroll area
DH,DL = row, column of lower right corner of scroll area

**Return Registers:**
none

**See also:** Tandy Function 72h, Standard Functions 06h and 07h, TopView INT 15h Function 12h Subfunction 05h (Chapter 44)

---

### INT 10h Function BF00h
### *SELECT EXTERNAL MONITOR*

*Compaq Portable Extensions*

**Purpose:** Specify that the external monitor become the active monitor.

**Registers at call:**
AX = BF00h

**Return Registers:**
none

**Details:** All registers are preserved and the internal monitor is blanked.

**See also:** Function BF01h

---

### INT 10h Function BF01h
### *SELECT INTERNAL MONITOR*

*Compaq Portable Extensions*

**Purpose:** Specify that the internal monitor become the active monitor.

**Registers at call:**
AX = BF01h

**Return Registers:**
none

**Details:** All registers are preserved and the external monitor is blanked.
**See also:** Function BF00h

### INT 10h Function BF02h
*Compaq Portable Extensions*
### *SET MASTER MODE OF CURRENT CONTROLLER*

**Registers at call:**
AX = BF02h
BH = master mode
  04h CGA
  05h EGA
  07h MDA

**Return Registers:**
none

**See also:** Function BF03h

### INT 10h Function BF03h
*Compaq Portable/Systempro*
### *GET ENVIRONMENT*

**Registers at call:**
AX = BF03h
BX = 0000h

**Return Registers:**
BH = active monitor
  00h external
  01h internal
BL = master mode
  00h switchable VDU not present
  04h CGA
  05h EGA
  07h MDA
  08h switchable LCD controller present
CH = 00h (reserved)
CL = switchable VDU mode supported
  bit 0 = CGA supported
  bits 1,2 = reserved (1)
  bit 3 = MDA supported
  bits 4-7 = reserved (1)
DH = internal monitor type (see Table 8-32)
DL = external monitor type (see Table 8-32)

**See also:** Functions BF00h through BF02h, standard Function 1Ah (above)

**8-32 Values for monitor type:**

| | |
|---|---|
| 00h | none |
| 01h | dual-mode monitor |
| 02h | 5153 RGB monitor |
| 03h | Compaq Color monitor |
| 04h | 640x400 flat panel |
| 05h | VGC monochrome |
| 06h | VGC color |
| 07h | LCD VGA |

### INT 10h Function BF04h
*Compaq Portable Extensions*
### *SET MODE SWITCH DELAY*

**Registers at call:**
AX = BF04h
BH = new state of delay
  (00h enabled, 01h disabled)

**Return Registers:**
none

**See also:** Function BF05h

### INT 10h Function BF05h
*Compaq Systempro Extensions*

***ENABLE/DISABLE DISPLAY***

**Registers at call:**
AX = BF05h
BH = new state of video
  00h off, 01h on

**Return Registers:**
none

**See also:** Function BF04h, standard Function 12h Subfunction 36h (above)

### INT 10h Function BF06h
*Compaq SLT/286*

***READ GRAY SCALE TABLE***

**Registers at call:**
AX = BF06h
CL = address to be read from gray scale table

**Return Registers:**
AL bits 3-0 = value read from gray scale table
CL = address to be read from gray scale table

**See also:** Function BF07h, standard Function 12h Subfunction 33h (above)

### INT 10h Function BF07h
*Compaq SLT/286*

***WRITE GRAY SCALE TABLE***

**Registers at call:**
AX = BF07h
CH = value to write to gray scale table
CL = address to be written to gray scale table

**Return Registers:**
none

**See also:** Functions BF06h and BF08h

### INT 10h Function BF08h
*Compaq SLT/286*

***WRITE COLOR MIX REGISTERS***

**Registers at call:**
AX = BF08h
CH = blue and green weights
  bits 7-4 green weight
  bits 3-0 blue weight
CL = red weight
  bits 7-4 unused
  bits 3-0 red weight

**Return Registers:**
none

**See also:** Function BF07h, standard Function 12h Subfunction 33h (above)

### INT 10h Function FFh
*Oak VGA BIOS v1.02+*

***SET EMULATION***

**Registers at call:**
AH = FFh
AL = emulation
  43h ('C') CGA emulation
  45h ('E') EGA emulation
  4Dh ('M') Hercules emulation
  56h ('V') VGA emulation
ES:DI -> signature string " Calamity "

**Return Registers:**
VGA switched to suggested mode

**Conflicts:** GO32 (Chapter 37), TopView (Chapter 44), Carbon Copy Plus (Chapter 58)

**See also:** Function 5F01h, Paradise Function 007Fh, standard Function 00h (above)

## EGA Register Interface Library

The EGA Register Interface Library (RIL) was introduced to overcome a major shortcoming of the EGA hardware (corrected in later adapters): write-only registers. For a mouse driver or other system software to be able to restore the state of the video display after making alterations, it must have access to the original video register contents, which on the EGA is only possible by maintaining a software copy. Programs which access the EGA registers through the RIL interface instead of directly programming the hardware allow system software to properly undo any temporary changes made by the system software.

The EGA RIL is provided by EGA.SYS (distributed with MS Windows 3.0+ and MS-DOS 5.0+), most mouse drivers, the OS/2 compatibility box, and others.

---

### INT 10h Function F0h
*EGA Register Interface Library*

### *READ ONE REGISTER*

**Registers at call:**
AH = F0h
BL = register number
BH = 00h
DX = group index (see Table 8-33)

**Return Registers:**
BL = data

**See also:** Functions F1h and F2h,INT 2Fh Function BC00h (see Chapter 43)

---

### INT 10h Function F1h
*EGA Register Interface Library*

### *WRITE ONE REGISTER*

**Registers at call:**
AH = F1h
DX = group index (see Table 8-33)
 if single register:
  BL = value to write
 otherwise:
  BL = register number
  BH = value to write

**Return Registers:**
BL = data

**See also:** Functions F0h and F3h, SOLLEX Function 7F05h (Chapter 9)

---

### INT 10h Function F2h
*EGA Register Interface Library*

### *READ REGISTER RANGE*

**Registers at call:**
AH = F2h
CH = starting register number
CL = Number of registers (>1)
DX = group index
 00h CRTC (3B4h mono modes, 3D4h color modes)
 08h Sequencer 3C4h
 10h Graphics Controller 3CEh
 18h Attribute Controller 3C0h
ES:BX -> buffer, CL bytes

**Return Registers:**
none

**See also:** Functions F0h and F3h

---

### INT 10h Function F3h
*EGA Register Interface Library*

### *WRITE REGISTER RANGE*

**Registers at call:**
AH = F3h
CH = starting register
CL = number of registers (2 or more)
DX = group index (see Function F2h)
ES:BX -> buffer, CL bytes

**Return Registers:**
none

**See also:** Functions F1h and F2h, SOLLEX Function 7F05h (Chapter 9)

---

### INT 10h Function F4h
### *READ REGISTER SET*

*EGA Register Interface Library*

**Registers at call:**
AH = F4h
CX = number of registers to read (2 or more)
ES:BX -> table of records (see Table 8-33)

**Return Registers:**
register values in table filled in

**See also:** Functions F0h, F2h, and F5h

**8-33 Format of entries in table of register records:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | group index |
| | | Pointer/data chips |
| | | 00h CRTC (3B4h mono modes, 3D4h color modes) |
| | | 08h Sequencer 3C4h |
| | | 10h Graphics Controller 3CEh |
| | | 18h Attribute Controller 3C0h |
| | | Single registers |
| | | 20h Miscellaneous Output register 3C2h |
| | | 28h Feature Control register (3BAh mono modes, 3DAh color) |
| | | 30h Graphics 1 Position register 3CCh |
| | | 38h Graphics 2 Position register 3CAh |
| 02h | BYTE | register number (0 for single registers) |
| 03h | BYTE | register value |

---

### INT 10h Function F5h
### *WRITE REGISTER SET*

*EGA Register Interface Library*

**Registers at call:**
AH = F5h
CX = number of registers to write (2 or more)
ES:BX -> table of records (see Table 8-33)

**Return Registers:**
none

**See also:** Functions F1h, F3h, and F4h; SOLLEX Function 7F05h (Chapter 9)

---

### INT 10h Function F6h
### *REVERT TO DEFAULT REGISTERS*

*EGA Register Interface Library*

**Registers at call:**
AH = F6h

**Return Registers:**
none

**See also:** Function F7h

---

### INT 10h Function F7h
### *DEFINE DEFAULT REGISTER TABLE*

*EGA Register Interface Library*

**Registers at call:**
AH = F7h
DX = port number (see Table 8-33)
ES:BX -> table of one-byte entries,
one byte to be written to each register

**Return Registers:**
none

**See also:** Function F6h

---

### INT 10h Function FAh
### *INTERROGATE DRIVER*

*EGA Register Interface Library*

**Registers at call:**
AH = FAh
BX = 0000h

**Return Registers:**
BX = 0000h if RIL not present
ES:BX -> EGA Register Interface version number, if present:

byte 0 = major release number
byte 1 = minor release number

**Conflicts:** FASTBUFF (Chapter 13)
**See also:** Function F6h, EGA.SYS INT 2Fh Function BC00h (Chapter 43)

## Hercules GRAFIX

---

### INT 10h Function 40h
### *SET GRAPHICS MODE*

Hercules GRAFIX

**Purpose:** Switch the Hercules Graphics Card into graphics mode (720x348).

| Registers at call: | Return Registers: |
|---|---|
| AH = 40h | none |

**See also:** Function 41h, standard Function 00h (above)

---

### INT 10h Function 41h
### *SET TEXT MODE*

Hercules GRAFIX

**Purpose:** Switch the Hercules Graphics Card into text mode.

| Registers at call: | Return Registers: |
|---|---|
| AH = 41h | none |

**See also:** Function 40h, standard Function 00h (above)

---

### INT 10h Function 42h
### *CLEAR CURRENT PAGE*

Hercules GRAFIX

| Registers at call: | Return Registers: |
|---|---|
| AH = 42h | none |

**See also:** Function 45h

---

### INT 10h Function 43h
### *SELECT DRAWING PAGE*

Hercules GRAFIX

**Purpose:** Specify which of the two pages of video memory is to be used for output.

| Registers at call: | Return Registers: |
|---|---|
| AH = 43h<br>AL = page number (0,1) | none |

**See also:** Functions 44h and 45h, standard Function 05h (above)

---

### INT 10h Function 44h
### *SELECT DRAWING FUNCTION*

Hercules GRAFIX

**Purpose:** Specify how graphics output will change the display.

| Registers at call: | Return Registers: |
|---|---|
| AH = 44h<br>AL = drawing function<br>00h clear pixels<br>01h set pixels<br>02h invert pixels | none |

**See also:** Functions 45h, 46h, 4Ch, and 4Dh

---

### INT 10h Function 45h
### *SELECT PAGE TO DISPLAY*

Hercules GRAFIX

**Purpose:** Specify which of the two pages of video memory is visible on screen.

| Registers at call: | Return Registers: |
|---|---|
| AH = 45h<br>AL = page number (0,1) | none |

**See also:** Functions 42h and 43h, standard Function 05h (above)

---

**INT 10h Function 46h** *Hercules GRAFIX*

***DRAW ONE PIXEL***

| Registers at call: | Return Registers: |
|---|---|
| AH = 46h<br>DI = x (0-720)<br>BP = y (0-347) | none |

**Details:** Function 44h determines the drawing operation and Function 43h which page to use.

**See also:** Functions 47h, 49h, 4Ch, and 4Dh; standard Function 0Ch (above)

---

**INT 10h Function 47h** *Hercules GRAFIX*

***FIND PIXEL VALUE***

| Registers at call: | Return Registers: |
|---|---|
| AH = 47h<br>DI = x (0-720)<br>BP = y (0-347) | AL = 00h pixel clear<br>AL = 01h pixel set |

**Details:** Function 43h specifies which page is used.

**See also:** Function 46h, standard Function 0Dh (above)

---

**INT 10h Function 48h** *Hercules GRAFIX*

***MOVE TO POINT***

**Purpose:** Specify the location from which to start the next graphics output.

| Registers at call: | Return Registers: |
|---|---|
| AH = 48h<br>DI = x (0-720)<br>BP = y (0-347) | none |

**See also:** Function 49h

---

**INT 10h Function 49h** *Hercules GRAFIX*

***DRAW TO POINT***

| Registers at call: | Return Registers: |
|---|---|
| AH = 49h<br>DI = x (0-720)<br>BP = y (0-347) | none |

**Details:** Function 48h or 49h specify the first point, Function 44h the drawing operation, and Function 43h the page to use.

**See also:** Functions 43h, 44h, 48h, 4Ch, and 4Dh

---

**INT 10h Function 4Ah** *Hercules GRAFIX*

***BLOCK FILL***

**Purpose:** Draw a solid rectangle of the specified size at the given location.

| Registers at call: | Return Registers: |
|---|---|
| AH = 4Ah<br>DI = x coordinate of lower left corner<br>BP = y coordinate of lower left corner<br>BX = height in pixels<br>CX = width in pixels | none |

**See also:** Function 4Eh

---

**INT 10h Function 4Bh** *Hercules GRAFIX*

***DISPLAY CHARACTER***

**Registers at call:**
AH = 4Bh
AL = character to display

DI = x (0-720)
BP = y (0-347)

**Return Registers:**
none

**Details:** Unlike the other BIOS character output functions, the character position is specified in pixels rather than rows and columns.
**Conflicts:** Frieze (Chapter 15)
**See also:** standard Functions 09h and 0Ah (above)

---

### INT 10h Function 4Ch
*Hercules GRAFIX*

### *DRAW ARC*

**Registers at call:**
AH = 4Ch
AL = quadrant (1 = upper right, 2 = upper left, etc)
DI = x coordinate of center
BP = y coordinate of center
BX = radius

**Return Registers:**
none

**See also:** Functions 49h and 4Dh

### INT 10h Function 4Dh
*Hercules GRAFIX*

### *DRAW CIRCLE*

**Registers at call:**
AH = 4Dh
DI = x coordinate of center
BP = y coordinate of center
BX = radius

**Return Registers:**
none

**See also:** Functions 49h and 4Ch

### INT 10h Function 4Eh
*Hercules GRAFIX*

### *FILL AREA*

**Purpose:** Fill a convex polygonal area bounded by a contiguous line of the opposite color with the border color.

**Registers at call:**
AH = 4Eh
DI = x coordinate of an interior point
BP = y coordinate of an interior point

**Return Registers:**
none

**Details:** The first fill makes the figure solid, the second erases it.
**See also:** Function 4Ah

## UltraVision

### INT 10h Function CC00h
*UltraVision*

### *GET STATUS (INSTALLATION CHECK)*

**Registers at call:**
AX = CC00h
SI = 0000h (if checking version)

**Return Registers:**
CX = ABCDh
AL = Ultravision extensions
    00h enabled
    FFh disabled
AH = card designator
BX:00F0h -> palette values (for compatibility with NEWFONT)
DX = support for high resolution modes
    00h not active

01h active
SI = UltraVision version number (v1.2+), high byte=major,low byte=minor
(unchanged for versions prior to 1.2)

**See also:** Functions CC01h and CC02h

---

### INT 10h Function CC01h
### *DISABLE EXTENSIONS*

*UltraVision*

**Registers at call:**
AX = CC01h

**Return Registers:**
none

**Details:** Subsequent BIOS calls will be passed through to the previous handler.

Calls to this function should be followed immediately by a mode set to restore the normal EGA/VGA state.

**See also:** Function CC02h

---

### INT 10h Function CC02h
### *ENABLE EXTENSIONS*

*UltraVision*

**Registers at call:**
AX = CC02h

**Return Registers:**
none

**Details:** Calls to this function should be followed immediately by a mode set to restore the previous UltraVision state.

**See also:** Function CC01h

---

### INT 10h Function CD00h
### *LOAD ULTRAVISION PALETTE*

*UltraVision*
*on color EGA and VGA only*

**Registers at call:**
AX = CD00h
CL = palette table number (01h-07h for v1.x, 01h-0Fh for v2+)
DS:DX -> 16-byte palette register list (colors for registers 00h-0Fh)

**Return Registers:**
none

**Details:** If palette locking is in effect for the current mode, the new colors will be displayed immediately; otherwise, the system reverts to the default palette. Palette table 0 is reserved for the default palette and cannot be set. UltraVision always sets the border color to black.

**See also:** Functions CD01h and CD02h

---

### INT 10h Function CD01h
### *SET PALETTE LOCKING STATUS*

*UltraVision*
*on color EGA and VGA only*

**Registers at call:**
AX = CD01h
CL = palette locking value
00h none
01h text modes only (02h,03h)
FFh all modes (all standard color text and graphics modes)

**Return Registers:**
none

**Details:** Intended for video modes with 16 or fewer colors.

**See also:** Functions CD00h and CD03h, standard Functions 1000h and 1002h (above)

---

### INT 10h Function CD02h
### *GET ULTRAVISION PALETTE*

*UltraVision*
*on EGA and VGA only*

**Registers at call:**
AX = CD02h

**Return Registers:**
CL = palette table number
DS:DX -> 17-byte palette register list (see Table 8-34)
DS:SI -> current font names table (see Tables 8-35 and 8-36)

**Details:** Only the font names are valid on monochrome EGA systems.

**See also:** Function CD00h, standard Function 1009h (above)

**8-34 Format of palette register list:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 16 BYTEs | colors for palette registers 00h through 0Fh |
| 10h | BYTE | border color |

**8-35 Format of current font names table (v2+):**

| *Offset* | *Size* | *standard EGA* | *HiRes EGA* | *VGA* |
|---|---|---|---|---|
| 00h | 8 BYTEs | N/A | F19 font | F20 font |
| 08h | 8 BYTEs | F14 font | F14 font | F14 font |
| 10h | 8 BYTEs | N/A | F11 font | F10 font |
| 18h | 8 BYTEs | F8 font | F8 font | F8 font |

**8-36 Format of current font names table (v1.x):**

| *Offset* | *Size* | *HiRes EGA* |
|---|---|---|
| 00h 8 | BYTEs | F19/F14 font |
| 08h 8 | BYTEs | F11/F8 font |

### INT 10h Function CD03h

***GET PALETTE LOCKING STATUS***

*UltraVision*
*on color EGA and VGA only*

**Registers at call:**
AX = CD03h

**Return Registers:**
CL = palette locking value
00h none
01h text modes only
FFh all modes

**See also:** Function CD01h

### INT 10h Function CD04h

***GET UltraVision TEXT MODE***

*UltraVision*
*on EGA and VGA only*

**Registers at call:**
AX = CD04h

**Return Registers:**
AL = mode number (see Table 8-37)

**See also:** Function CC00h, Function CDh, standard Function 0Fh (above)

**8-37 Values for video mode number:**

| | | | |
|---|---|---|---|
| 11h | 80x25 | 23h | 107x34, 108x36 |
| 12h | 80x43, 80x50 | 24h | 108x60, 108x63 |
| 13h | 80x34, 80x36 | 31h | 120x25 |
| 14h | 80x60, 80x63 | 32h | 120x43, 120x50 |
| 19h | 94x25 | 33h | 132x25 |
| 1Ah | 94x43, 94x50 | 34h | 132x44, 132x50 |
| 1Bh | 94x36 | 39h | 120x36 |
| 1Ch | 94x63 | 3Ah | 120x63 |
| 21h | 108x25 | 3Bh | 132x36 |
| 22h | 108x43, 108x50 | 3Ch | 132x60 |

**See also:** Tables 8-2, 8-19, 8-27, 9-4, and 9-5

### INT 10h Function CD05h

***SET CURSOR TYPE***

*UltraVision*
*on EGA and VGA only*

**Purpose:** This function sets the default cursor type for text-based programs.

**Registers at call:**
AX = CD05h
CL = cursor type (see Table 8-38)

**Return Registers:**
none

**See also:** Function CD06h, standard Function 01h (above)

**8-38 Values for cursor type:**

| | |
|---|---|
| 00h | line cursor |
| FFh | box cursor |

---

### INT 10h Function CD06h
### *GET CURSOR TYPE*

*UltraVision*
*on EGA and VGA only*

**Registers at call:**
AX = CD06h

**Return Registers:**
CL = current cursor type (see Table 8-38)

**See also:** Function CD05h, standard Function 03h (above)

---

### INT 10h Function CD07h
### *SET UNDERLINE STATUS*

*UltraVision v1.2+*
*on EGA and VGA only*

**Registers at call:**
AX = CD07h
CL = hardware underline status (see Table 8-39)
BL = foreground color for normal text (FFh = current)
BH = foreground color for bright text (FFh = current)

**Return Registers:**
CL = hardware underline status
BL = current foreground color for normal text
BH = current foreground color for bright text

**Details:** When underline or strikeout is enabled in color text modes, the specified colors will be assigned temporarily to colors 01h and 09h, allowing affected text to match non-underlined text. The color remapping uses values from the current onscreen palette regardless of the palette locking status (see Function CD01h).Specify the standard colors (BL=01h,BH=09h) to enable underline or strikeout without color remapping.

**See also:** Function CD08h

**8-39 Values for hardware underline status:**

| | |
|---|---|
| 00h | off (color systems only) |
| 01h | underline below characters |
| 02h | strike through characters |

---

### INT 10h Function CD08h
### *GET UNDERLINE STATUS*

*UltraVision v1.2+*
*on EGA and VGA only*

**Registers at call:**
AX = CD08h

**Return Registers:**
CL = hardware underline status (see Table 8-39)
BL = foreground color for normal text
BH = foreground color for bright text

**Details:** Only CL is valid on monochrome EGA systems.

**See also:** Function CD07h

---

### INT 10h Function CD10h
### *LOAD USER FONT*

*UltraVision*
*on EGA and VGA only*

**Purpose:** Loads the designated characters into UltraVision's resident font area.

**Registers at call:**
AX = CD10h
BH = bytes per character (08h,0Ah,0Bh,0Eh,13h,14h)
CX = ABCDh to load 9xN alternate font (v2+) else number of characters to load
DX = character offset into font table

DS:SI -> 8-byte ASCII font name
ES:BP -> font definitions

**Return Registers:**
AX = FFFFh if invalid font parameters

**Details:** Calls to this function should be followed by a video mode set to reload the character generator.

**See also:** Standard Function 1100h (above)

---

### INT 10h Function CDh
### *SET ULTRAVISION TEXT MODE*

*UltraVision*
*on EGA and VGA only*

**Registers at call:**
AH = CDh
AL = text mode number (see Table 8-37)

**Return Registers:**
AX = CDCDh if invalid mode

**See also:** Function CD04h

## Miscellaneous Display Drivers

---

### INT 10h Function 0F56h, Subfunction 4756h
### *DRIVER API*

*VUIMAGE DISPLAY DRIVER*

**Registers at call:**
AX = 0F56h
BX = 4756h ('GV')
CX = 4944h ('IA')
DL = function

**Return Registers:**

01h installation check
AX = 5649h ('VI')
BX = 4443h ('DC')
CX = 5647h ('VG')
DH = 01h

02h get first video mode's parameters
AX = BIOS mode number
BX = width in pixels
CX = height in pixels
DX = number of colors

03h get next video mode's parameters
as above

04h *display line*
ES:DI -> record (see Table 8-40)
*other registers unknown*
*unknown*

**Program:** VUIMAGE is a shareware GIF/TIFF image viewer by Offe Enterprises

**Details:** The use of TSR display drivers for VUIMAGE was discontinued after v2.20.

**Conflicts:** Standard Function 0Fh (above)

**8-40 Format of record for function 04h:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | row number |
| 02h | WORD | starting column |
| 04h | WORD | ending column |

**Note**: additional fields, if any, unknown

---

### INT 10h Function 10h
### *SET WINDOW COORDINATES*

*BIOS Window Extension v1.1*

**Registers at call:**
AH = 10h

CH,CL = row,column of upper left corner of window
DH,DL = row,column of lower right corner of window

**Return Registers:**
AL = status
  00h successful
  01h failed
AH destroyed

**Program:** BWE is a TSR by John J. Seal published in the May 1986 Dr. Dobb's Journal.
**Details:** When a window has been set, all output via Function 0Eh is restricted to the specified window.
**Conflicts:** Standard Functions 10xxh (above), Tseng Functions 10Fxh (Chapter 9)
**See also:** Functions 11h and 12h

---

### INT 10h Function 11h

*BIOS Window Extension v1.1*

### *GET WINDOW COORDINATES*

**Purpose:** Determine the portion of the display to which output is restricted
**Registers at call:**
AH = 11h

**Return Registers:**
CH,CL = row,column of upper left corner
DH,DL = row,column of lower right corner

**Conflicts:** Standard Functions 11xxh (above), Realtek RTVGA (above), M10_SCR below
**See also:** Functions 10h and 12h

---

### INT 10h Function 1130h, Subfunction 4D4Fh

*M10_SCR.COM v3.5+*

### *INSTALLATION CHECK*

**Registers at call:**
AX = 1130h
BX = 4D4Fh

**Return Registers:**
CX = 4F4Dh if installed
  ES:BP -> M10_SCR INT 10h handler

**Program:** M10_SCR is a shareware extended text mode and font driver by I.V. Morozov
**Conflicts:** Standard Function 1130h (above), BIOS Window Extension
**See also:** M16_KBD INT 16h Function 4D4Fh (Chapter 13)

---

### INT 10h Function 12h

*BIOS Window Extension v1.1*

### *GET BLANKING ATTRIBUTE*

**Registers at call:**
AH = 12h

**Return Registers:**
BH = attribute to use on blanked lines when scrolling

**Program:** BWE is a TSR by John J. Seal published in the May 1986 Dr. Dobb's Journal.
**Conflicts:** Standard Function 12h (above), Cirrus Logic BIOS (Chapter 9), Trident BIOS (Chapter 9)
**See also:** Functions 11h and 12h

---

### INT 10h Function 50h

*SCROLOCK.COM*

### *INSTALLATION CHECK*

**Registers at call:**
AH = 50h

**Return Registers:**
BX = 1954h if installed
  AL = 00h if inactive, nonzero if active

**Program:** SCROLOCK is a utility supplied with System Enhancement Associates' ARC.
**Conflicts:** SCREENR below, AX PC (Chapter 6)
**See also:** Function 51h

---

### INT 10h Function 5049h

*SCREENR v1.55+*

### *API*

**Registers at call:**
AX = 5049h ('PI')
BX = function
  0000h installation check
  0001h lock mode
  0002h unlock mode

0003h lock palette
0004h unlock palette

**Return Registers:**
AX = 0000h if installed
BX = TSR version (BH=major,BL=minor)
CL = mode locking status
00h mode not locked
01h mode locked: INT 10h Function 00h disabled
CH = palette locking status
00h palette not locked
01h palette locked, Functions 1000h, 1001h, 1002h, 1010h, and 1012h are disabled

**Program:** SCREENR is a TSR supplied with Patrick Ibbetson's SCREEN display utility.
**Conflicts:** SCROLOCK above, AX PC (Chapter 6)

---

### INT 10h Function 51h
*SCROLOCK.COM*

### *ENABLE/DISABLE*

**Registers at call:**
AH = 51h
AL = state
00h disable
nonzero enable

**Return Registers:**
none

**Program:** SCROLOCK is a utility supplied with System Enhancement Associates' ARC.
**Conflicts:** AX PC (Chapter 6)
**See also:** Function 50h

---

### INT 10h Function 53h
*Show Partner F/X v3.6*

### *START PRESENTATION*

**Registers at call:**
AH = 53h
DS:DX -> ASCIZ name of presentation file (no path, extension forced to .PR2)

**Return Registers:**
*unknown*

**See also:** Function 55h

---

### INT 10h Function 55h
*Show Partner F/X v3.6*

### *UNINSTALL*

**Registers at call:**
AH = 55h

**Return Registers:**
FXSHOW.EXE removed from memory

**Conflicts:** ATI SMS.COM below
**See also:** Function 53h

---

### INT 10h Function 5555h
*ATI SMS.COM*

### *INSTALLATION CHECK*

**Purpose:** Determine whether the ATI EGA/VGA Wonder Super Switch TSR is installed.

**Registers at call:**
AX = 5555h

**Return Registers:**
AX = AAAAh if installed
BX:CX -> *unknown* routine in SMS.COM resident portion
-> data area in VCONFIG

**Program:** Super Switch (SMS.COM) is a video mode switch program supplied with ATI EGA Wonder. It also maps video mode 08h to 27h or 23h, as appropriate.
**Conflicts:** Show Partner F/X above
**See also:** Standard Function 00h (above), SCRNSAV2 INT 2Fh Function 6400h (Chapter 14)

### 8-41 Format of data area:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | original INT 09h vector |
| 04h | DWORD | original INT 10h vector |
| 08h | DWORD | original INT 1Ch vector |
| 0Ch | WORD | screen saver state, 0=off, 1=on |
| 0Eh | WORD | blanking interval in clock ticks |

---

## INT 10h Function 6A00h

*Direct Graphics Interface Standard*

### *INQUIRE AVAILABLE DEVICES*

**Registers at call:**
AX = 6A00h
BX = 0000h
CX = 0000h
DX = buffer length (may be 0)
ES:DI -> buffer

**Return Registers:**
BX = number of bytes stored in buffer
CX = bytes required for all descriptions (0 if no DGIS)
ES:DI buffer filled with descriptions and addresses of DGIS-compatible display(s) and printer(s)

**See also:** Function 6A02h

---

## INT 10h Function 6A01h

*Direct Graphics Interface Standard*

### *REDIRECT CHARACTER OUTPUT*

**Registers at call:**
AX = 6A01h
CX = 0000h
ES:DI = address of device to send INT 10h output to

**Return Registers:**
CX = status
 0000h output could not be redirected
 else INT 10h output now routed to requested display

**See also:** Function 6A02h

---

## INT 10h Function 6A02h

*Direct Graphics Interface Standard*

### *INQUIRE INT 10h OUTPUT DEVICE*

**Registers at call:**
AX = 6A02h
ES:DI = 0000h:0000h

**Return Registers:**
ES:DI = status
 0000h:0000h if current display is non-DGIS
 else address of the current DGIS INT 10h display

**See also:** Functions 6A00h and 6A01h

---

## INT 10h Function BFA0h

*Compaq ADAPT.COM*

### *INSTALLATION CHECK*

**Registers at call:**
AX = BFA0h
BX = ADADh

**Return Registers:**
AX = BDBDh if newer ADAPT.COM installed
 BX = BCD version (BH = major, BL = minor)
 CL = *unknown*
 CH = *unknown*
 DL = *unknown*
AX = ADADh if older version of ADAPT.COM installed

**Program:** ADAPT is Compaq's Advanced Display Attribute Programming Tool, an optionally-resident utility for setting display colors and cursor size; when resident, it also includes a screen blanker.

**See also:** Functions BFA1h, BFA2h, and DFA5h

---

### INT 10h Function BFA1h

*Compaq ADAPT.COM*

***GET UNKNOWN VALUES***

**Registers at call:**
AX = BFA1h

**Return Registers:**
AX = BDBEh if supported
CH = current value of *unknown item*
DL = current value of *unknown item*

**See also:** Functions BFA0h and BFA2h

---

### INT 10h Function BFA2h

*Compaq ADAPT.COM*

***SET UNKNOWN VALUES***

**Registers at call:**
AX = BFA2h
CH = new value for *unknown item*
DL = new value for *unknown item*

**Return Registers:**
AX = BDBEh if supported

**See also:** Functions BFA0h and BFA1h

---

### INT 10h Function D5h

*Netroom SCRNCLK*

***UNDETERMINED FUNCTION***

**Registers at call:**
AH = D5h
*other registers, if any, unknown*

**Return Registers:**
*unknown*

**Program:** SCRNCLK is a “ cloaked ” screen accelerator included with Helix’s Netroom version 3.

---

### INT 10h Function DFA5h

*Compaq ADAPT.COM (undocumented)*

***GET UNKNOWN DATA AREA***

**Registers at call:**
AX = DFA5h

**Return Registers:**
AX = BDBFh if supported
ES:DI -> *unknown* data area
BX = ES

**See also:** Function BFA0h

---

### INT 10h Function EFh

*MSHERC.COM*

***GET VIDEO ADAPTER TYPE AND MODE***

**Registers at call:**
AH = EFh

**Return Registers:**
DL = video adapter type
  00h original Hercules
  01h Hercules Plus
  02h *Hercules InColor*
DH = memory mode byte
  01h “ half ” mode
  03h “ full ” mode

**Program:** MSHERC.COM is a support program for the Microsoft Quick languages which makes their graphics libraries compatible with a Hercules card by adding video modes 08h and 88h, and supporting text in the new graphics modes.

**Details:** While in mode 08h or 88h, INT 10h supports the Hercules card much like a CGA. MSHERC performs an installation check by setting DL=FFh and testing whether it has been changed on return.

---

### INT 14h Function 8100h

*VIDEO FOSSIL*

***RETURN VFOSSIL INFORMATION***

**Registers at call:**
AX = 8100h
ES:DI -> buffer for VFOSSIL information (see Table 8-42)

**Return Registers:**
AX = 1954h if installed

**See also:** Function 8101h, FOSSIL Function 7Eh (Chapter 12)

### 8-42 Format of VFOSSIL information:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | size of information in bytes, including this field |
| 02h | WORD | VFOSSIL major version |
| 04h | WORD | VFOSSIL revision level |
| 06h | WORD | highest VFOSSIL application function supported |

---

## INT 14h Function 8101h

*VIDEO FOSSIL*

### *OPEN VFOSSIL*

**Registers at call:**
AX = 8101h
ES:DI -> buffer for application function table (see Table 8-43)
CX = length of buffer in bytes

**Return Registers:**
AX = 1954h if installed
BH = highest VFOSSIL application function supported

**Details:** The number of initialized pointers in the application function table will never exceed CX/4; if the buffer is large enough, BH+1 pointers will be initialized.

**See also:** Functions 8100h and 8102h

### 8-43 Format of application function table:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | -> function to query current video mode (VioGetMode) |
| 04h | DWORD | -> function to set video mode (VioSetMode) |
| 08h | DWORD | -> function to query hardware config (VioGetConfig) |
| 0Ch | DWORD | -> function to write data in TTY mode (VioWrtTTY) |
| 10h | DWORD | -> function to get current ANSI state (VioGetANSI) |
| 14h | DWORD | -> function to set new ANSI state (VioSetANSI) |
| 18h | DWORD | -> function to get curr cursor position (VioGetCurPos) |
| 1Ch | DWORD | -> function to set cursor position (VioSetCurPos) |
| 20h | DWORD | -> function to get cursor shape (VioGetCurType) |
| 24h | DWORD | -> function to set cursor shape (VioSetCurType) |
| 28h | DWORD | -> function to scroll screen up (VioScrollUp) |
| 2Ch | DWORD | -> function to scroll screen down (VioScrollDn) |
| 30h | DWORD | -> function to read cell string from screen (VioReadCellStr) |
| 34h | DWORD | -> function to read char string from screen (VioReadCharStr) |
| 38h | DWORD | -> function to write a cell string (VioWrtCellStr) |
| 3Ch | DWORD | -> function to write char string, leaving attr (VioWrtCharStr) |
| 40h | DWORD | -> function to write char string,const attr (VioWrtCharStrAttr) |
| 44h | DWORD | -> function to replicate an attribute (VioWrtNAttr) |
| 48h | DWORD | -> function to replicate a cell (VioWrtNCell) |
| 4Ch | DWORD | -> function to replicate a character (VioWrtNChar) |

### 8-44 Format of video mode data structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | length of structure including this field |
| 02h | BYTE | mode characteristics<br>bit 0: clear if MDA, set otherwise<br>bit 1: graphics mode<br>bit 2: color disabled (black-and-white) |
| 03h | BYTE | number of colors supported (1=2 colors, 4=16 colors, etc) |
| 04h | WORD | number of text columns |
| 06h | WORD | number of text rows |
| 08h | WORD | reserved |

| | | |
|---|---|---|
| 0Ah | WORD | reserved |
| 0Ch | DWORD | reserved |

**8-45 Format of video configuration data:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | structure length including this field |
| 02h | WORD | adapter type |
| | | 00h monochrome/printer |
| | | 01h CGA |
| | | 02h EGA |
| | | 03h VGA |
| | | 07h 8514/A |
| 04h | WORD | display type |
| | | 00h monochrome |
| | | 01h color |
| | | 02h enhanced color |
| | | 09h 8514 |
| 06h | DWORD | adapter memory size |

**8-46 Format of cursor type record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | cursor start line |
| 02h | WORD | cursor end line |
| 04h | WORD | cursor width (always 01h) |
| 06h | WORD | cursor attribute (FFFFh = hidden) |

**8-47 VioGetMode calling convention:**

| **Registers at call:** | | | **Return Registers:** |
|---|---|---|---|
| STACK: | WORD | VIO handle (must be 00h) | AX = error code (00h, 74h, 17Eh, 1B4h) (see Table 8-48) |
| | DWORD | pointer to video mode data structure (see Table 8-44) | |

**8-48 Values for error code:**

| | |
|---|---|
| 0000h | successful |
| 0074h | internal VIO failure |
| 0163h | unsupported mode |
| 0166h | invalid row value |
| 0167h | invalid column value |
| 017Eh | buffer too small |
| 01A5h | invalid VIO parameter |
| 01B4h | invalid VIO handle |

**8-49 VioSetMode calling convention:**

| **Registers at call:** | | | **Return Registers:** |
|---|---|---|---|
| STACK: | WORD | VIO handle (must be 00h) | AX = error code (00h,74h,163h,17Eh,1A5h,1B4h) (see Table 8-48) |
| | DWORD | pointer to video mode data structure (see Table 8-44) | |

### 8-50 VioGetConfig calling convention:

**Registers at call:**

STACK: WORD VIO handle (must be 00h)
DWORD pointer to video configuration data buffer (see Table 8-45)

**Return Registers:**

AX = error code (00h,74h,17Eh,1B4h) (see Table 8-48)

### 8-51 VioWrtTTY calling convention:

**Registers at call:**

STACK: WORD VIO handle (must be 00h)
WORD length of string
DWORD pointer to character string to be written to screen

**Return Registers:**

AX = error code (00h,74h,1B4h) (see Table 8-48)

**Details:** The write wraps at end of line and terminates if it reaches end of the screen. In ANSI mode, ANSI control sequences are interpreted, and this function is not required to be reentrant; in non-ANSI mode, the function is reentrant and may be called from within an MS-DOS function call.

### 8-52 VioGetANSI calling convention:

**Registers at call:**

STACK: WORD VIO handle (must be 00h)
DWORD pointer to WORD which will be set to 00h if ANSI is off or 01h if ANSI is on

**Return Registers:**

AX = error code (00h,74h,1B4h) (see Table 8-48)

### 8-53 VioSetANSI calling convention:

**Registers at call:**

STACK: WORD VIO handle (must be 00h)
DWORD pointer to WORD indicating new state of ANSI (00h off, 01h on)

**Return Registers:**

AX = error code (00h,74h,1A4h,1B4h) (see Table 8-48).

### 8-54 VioGetCurPos calling convention:

**Registers at call:**

STACK: WORD VIO handle (must be 00h)
DWORD pointer to WORD to hold current cursor column (0-based)
DWORD pointer to WORD to hold current cursor row (0-based)

**Return Registers:**

AX = error code (00h,74h,1B4h) (see Table 8-48)

### 8-55 VioSetCurPos calling convention:

**Registers at call:**

STACK: WORD VIO handle (must be 00h)
WORD cursor column
WORD cursor row

**Return Registers:**

AX = error code (00h,74h,166h,167h,1B4h) (see Table 8-48)

**Details:** If either coordinate is invalid, the cursor is not moved.

### 8-56 VioGetCurType calling convention:

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| STACK: | WORD | VIO handle (must be 00h) | AX = error code (00h,74h,1B4h) (see Table 8-48) |
| | DWORD | pointer to cursor type record (see Table 8-46) | |

### 8-57 VioSetCurType calling convention:

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| STACK: | WORD | VIO handle (must be 00h) | AX = error code (00h,74h,1A4h,1B4h) (see Table 8-48) |
| | DWORD | pointer to cursor type record (see Table 8-46) | |

### 8-58 VioScrollUp calling convention:

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| STACK: | WORD | VIO handle (must be 00h) | AX = error code (00h,74h,166h,167h,1B4h) (see Table 8-48) |
| | DWORD | pointer to char/attr cell for filling emptied rows | |
| | WORD | number or rows to scroll (FFFFh = clear area) | |
| | WORD | right column of scroll area | |
| | WORD | bottom row of scroll area | |
| | WORD | left column of scroll area | |
| | WORD | top row of scroll area | |

### 8-59 VioScrollDn calling convention:

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| STACK: | WORD | VIO handle (must be 00h) | AX = error code (00h,74h,166h,167h,1B4h) (see Table 8-48) |
| | DWORD | pointer to char/attr cell for filling emptied rows | |
| | WORD | number or rows to scroll (FFFFh = clear area) | |
| | WORD | right column of scroll area | |
| | WORD | bottom row of scroll area | |
| | WORD | left column of scroll area | |
| | WORD | top row of scroll area | |

### 8-60 VioReadCellStr calling convention:

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| STACK: | WORD | VIO handle (must be 00h) | AX = error code (00h,74h,166h,167h,1B4h) (see Table 8-48) |
| | WORD | column at which to start | |
| r | | eading | |
| | WORD | row at which to start reading | |
| | DWORD | pointer to WORD containing length of buffer in bytes on return, the WORD contains the umber of bytes actually read | |
| | DWORD | pointer to buffer for cell string | |

### 8-61 VioReadCharStr calling convention:

**Registers at call:**

STACK: WORD VIO handle (must be 00h)
WORD column at which to start reading
WORD row at which to start reading
DWORD pointer to WORD containing length of buffer in bytes on return, the WORD contains the number of bytes actually read
DWORD pointer to buffer for character string

**Return Registers:**

AX = error code (00h,74h,166h,167h,1B4h) (see Table 8-48)

### 8-62 VioWrtCellStr calling convention:

**Registers at call:**

STACK: WORD VIO handle (must be 00h)
WORD column at which to start writing
WORD row at which to start writing
WORD length of cell string in bytes
DWORD pointer to cell string to write

**Return Registers:**

AX = error code (00h,74h,166h,167h,1B4h) (see Table 8-48)

**Details:** Write wraps at end of line and terminates if it reaches end of screen.

### 8-63 VioWrtCharStr calling convention:

**Registers at call:**

STACK: WORD VIO handle (must be 00h)
WORD column at which to start writing
WORD row at which to start writing
WORD length of character string
DWORD pointer to character string to write

**Return Registers:**

AX = error code (00h,74h,166h,167h,1B4h) (see Table 8-48)

**Details:** Write wraps at end of line and terminates if it reaches end of screen.

### 8-64 VioWrtCharStrAttr calling convention:

**Registers at call:**

STACK: WORD VIO handle (must be 00h)
DWORD pointer to attribute to be applied to each character
WORD column at which to start writing
WORD row at which to start writing
WORD length of character string
DWORD pointer to character string to write

**Return Registers:**

AX = error code (00h,74h,166h,167h,1B4h) (see Table 8-48)

**Details:** Write wraps at end of line and terminates if it reaches end of screen.

### 8-65 VioWrtNAttr calling convention:

**Registers at call:**

| | | |
|---|---|---|
| STACK: | WORD | VIO handle (must be 00h) |
| | WORD | column at which to start writing |
| | WORD | row at which to start writing |
| | WORD | number of times to write attribute |
| | DWORD | pointer to display attribute to replicate |

**Return Registers:**
AX = error code (00h,74h,166h,167h,1B4h) (see Table 8-48)

**Details:** Write wraps at end of line and terminates if it reaches end of screen.

### 8-66 VioWrtNCell calling convention:

**Registers at call:**

| | | |
|---|---|---|
| STACK: | WORD | VIO handle (must be 00h) |
| | WORD | column at which to start writing |
| | WORD | row at which to start writing |
| | WORD | number of times to write cell |
| | DWORD | pointer to cell to replicate |

**Return Registers:**
AX = error code (00h,74h,166h,167h,1B4h) (see Table 8-48)

**Details:** Write wraps at end of line and terminates if it reaches end of screen.

### 8-67 VioWrtNChar calling convention:

**Registers at call:**

| | | |
|---|---|---|
| STACK: | WORD | VIO handle (must be 00h) |
| | WORD | column at which to start writing |
| | WORD | row at which to start writing |
| | WORD | number of times to write character |
| | DWORD | pointer to character to replicate |

**Return Registers:**
AX = error code (00h,74h,166h,167h,1B4h) (see Table 8-48)

**Details:** Write wraps at end of line and terminates if it reaches end of screen.

---

## INT 14h Function 8102h

*VIDEO FOSSIL*

### *CLOSE VFOSSIL*

**Registers at call:**
AX = 8102h

**Return Registers:**
AX = 1954h

**Details:** Terminates all operations; after this call, the video FOSSIL may either be removed from memory or reinitialized.

**See also:** Functions 8101h and 8103h

---

## INT 14h Function 8103h

*VIDEO FOSSIL*

### *UNINSTALL*

**Registers at call:**
AX = 8103h

**Return Registers:**
AX = 1954h

**Details:** This is an extension to the VFOSSIL spec by Bob Hartman's VFOS_IBM.

---

### INT 16h Function CA00h, Subfunction 6570h

*CtrlAlt Associates EGAPAL.COM v1.00*

***INSTALLATION CHECK***

**Registers at call:**
AX = CA00h
BX = 6570h ( " ep " )
CX = 6570h

**Return Registers:**
AX = CAFFh if installed
BX = segment of resident code
CX = *unknown* (0090h)

**Program:** EGAPAL is a TSR supplied with STACKEY which makes EGA palette settings permanent across mode switches.

**See also:** Function CA00h Subfunction 7670h

---

### INT 16h Function CA00h, Subfunction 7670h

*CtrlAlt Associates VGAPAL.COM v1.00*

***INSTALLATION CHECK***

**Registers at call:**
AX = CA00h
BX = 7670h ( " vp " )
CX = 7670h

**Return Registers:**
AX = CAFFh if installed
BX = segment of resident code
CX = *unknown* (0090h)

**Program:** VGAPAL is a TSR supplied with STACKEY which makes VGA palette settings permanent across mode switches.

**See also:** Function CA00h Subfunction 6570h

---

### INT 17h Function 60h

*FLASHUP.COM*

***INSTALLATION CHECK***

**Registers at call:**
AH = 60h

**Return Registers:**
AL = 60h
DX = CS of resident code

**Details:** FLASHUP.COM is part of Flash-Up Windows by The Software Bottling Co.

FLASHUP also hooks INT 10h and receives commands via INT 10h Functions 09h/0Ah consisting of an 80h followed by the actual command.

**See also:** Standard INT 10h Functions 09h and 0Ah (above)

---

### INT 17h Function 61h

*SPEEDSCR.COM*

***INSTALLATION CHECK***

**Registers at call:**
AH = 61h

**Return Registers:**
AL = 61h
DX = CS of resident code

**Program:** SPEEDSCR.COM is by The Software Bottling Co.

---

### INT 21h Function 4402h

*PGS1600.DEV*

***IOCTL - GET CONFIGURATION INFO***

**Registers at call:**
AX = 4402h
BX = file handle for device " PGS1600$ "
CX = 0018h (size of buffer)
DS:DX -> buffer for configuration info (see Table 8-68)

**Return Registers:**
CF clear if successful
buffer filled
AX = number of bytes actually copied
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Program:** PGS1600.DEV is a device driver for the Cornerstone Technology PG1600 display adapter, which provides a 1600x1200 monochrome display as well as one of two emulations, MDA or CGA.

**See also:** PGS1600 INT 21h Function 4403h

### 8-68 Format of configuration information:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | version (high byte = major, low byte = minor) |
| 02h | WORD | board initialisation mode |
| 04h | WORD | board I/O address |
| | | 03D0h CGA emulation |
| | | 03B0h MDA emulation |
| | | 0390h no emulation |
| | | 0350h no emulation, alternate |
| 06h | WORD | emulation buffer segment |
| | | B800h CGA emulation |
| | | B000h MDA emulation |
| | | 0000h no emulation |
| 08h | WORD | PG1600 graphics buffer segment |
| 0Ah | WORD | number of bytes between consecutive graphic rows |
| 0Ch | WORD | horizontal pixel size |
| 0Eh | WORD | vertical pixel size |
| 10h | WORD | horizontal dots per inch |
| 12h | WORD | vertical dots per inch |
| 14h | WORD | graphics buffer bits per pixel |
| 16h | WORD | monitor bits per pixel |

## INT 21h Function 4402h

*Compaq AG1024.SYS*

### *RGDI - GET DRIVER LOCATION*

**Registers at call:**

AX = 4402h

BX = file handle for device " $$$$RGDI "

CX = 0006h (size of returned data)

DS:DX -> location record (see Table 8-69)

**Return Registers:**

CF clear if successful

buffer filled

CF set on error

AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Program:** AG1024.SYS is a device driver for the Advanced Graphics 1024 adapter

**See also:** RGDI INT 21h Function 4403h

### 8-69 Format of location record:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | signature 55AAh |
| 02h | WORD | segment of *unknown item* |
| 04h | WORD | segment of device driver's code |

## INT 21h Function 4403h

*PGS1600.DEV*

### *IOCTL - SET CONFIGURATION*

**Registers at call:**

AX = 4403h

BX = file handle for device " PGS1600$ "

CX = 0018h (size of buffer)

DS:DX -> configuration buffer (see PGS1600 Function 4402h)

**Return Registers:**

CF clear if successful

AX = number of bytes actually written

CF set on error

AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Program:** PGS1600.DEV is a device driver for the Cornerstone Technology PG1600 display adapter, which provides a 1600x1200 monochrome display as well as one of two emulations, MDA or CGA.

**See also:** PGS1600 INT 21h Function 4402h

### INT 21h Function 4403h

*Compaq AG1024.SYS*

### *CPQ_MGES - IOCTL OUTPUT*

**Registers at call:**
AX = 4403h
BX = file handle referencing device " CPQ_MGES "
DS:DX -> request packet (see Table 8-70)
CX ignored

**Return Registers:**
CF clear if successful
  AX destroyed
  data buffer filled (if applicable)
  first word of request packet set to number of bytes of data available (amount returned is smaller of this and requested amount)
CF set on error
  AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Program:** AG1024.SYS is a device driver for the Advanced Graphics 1024 adapter
**See also:** RGDI INT 21h Function 4403h

**8-70 Format of request packet:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | function |
| | | 0000h get *unknown* (26h bytes) |
| | | 0001h get *unknown* (36h bytes) |
| | | 0002h set *unknown* (same as returned by function 0001h) |
| | | 0003h get *unknown* (6 bytes) |
| | | 0004h get *unknown* |
| | | 0005h get *unknown* |
| | | 0006h get *unknown* (10h bytes) |
| | | 0007h set *unknown* (same as returned by function 0006h) |
| *functions 00h-03h,06h,07h* | | |
| 02h | WORD | size of data buffer |
| 04h | DWORD | -> buffer for function's data |
| *functions 04h,05h* | | |
| 02h | WORD | *unknown* |
| 04h | WORD | size of data buffer |
| 06h | DWORD | -> buffer to receive data |

### INT 21h Function 4403h

*Compaq AG1024.SYS*

### *RGDI - IOCTL OUTPUT*

**Registers at call:**
AX = 4403h
BX = file handle referencing device " $$$$RGDI "
DS:DX -> request packet (see Table 8-71)
CX ignored

**Return Registers:**
CF clear if successful
  AX destroyed
  data buffer filled (if applicable)
  first word of request packet set to number of bytes of data available (amount returned is smaller of this and requested amount)
CF set on error
  AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Program:** AG1024.SYS is a device driver for the Advanced Graphics 1024 adapter
**See also:** RGDI INT 21h Function 4402h, CPQ_MGES INT 21h Function 4403h

### 8-71 Format of request packet:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | function |
| | | 0000h get entry points |
| | | 0001h get *unknown* |
| 02h | DWORD | address of buffer for returned data |

---

## INT 2Fh Function 1A00h, Subfunction 414Eh
*ANSIPLUS.SYS*

### *INSTALLATION CHECK*

**Registers at call:**
AX = 1A00h
BX = 414Eh ('AN')
CX = 5349h ('SI')
DX = 2B2Bh ('++')

**Return Registers:**
AL = FFh if installed
CF clear
ES:BX -> INT 29h entry point

**Program:** ANSIPLUS.SYS is a CON device driver by Kristofer Sweger which replaces the normal ANSI.SYS with a more powerful version with many additional features.

**Details:** ANSIPLUS also identifies itself as ANSI.SYS if BX,CX, or DX differ from the magic values above.

---

## INT 2Fh Function 1A00h, Subfunction 4156h
*AVATAR.SYS*

### *INSTALLATION CHECK*

**Registers at call:**
AX = 1A00h
BX = 4156h ('AV')
CX = 4154h ('AT')
DX = 4152h ('AR')

**Return Registers:**
AL = FFh if installed
CF clear
BX = AVATAR protocol level supported
CX = driver type
  0000h AVATAR.SYS
  4456h DVAVATAR.COM inside DESQview window
DX = 0016h

**Program:** AVATAR.SYS is a CON replacement by George Adam Stanislav which interprets AVATAR command codes in the same way that ANSI interprets ANSI command codes

**Details:** AVATAR also identifies itself as ANSI.SYS if BX, CX, or DX differ from the magic values.

---

## INT 2Fh Function 1A21h
*AVATAR.SYS*

### *SET DRIVER STATE*

**Registers at call:**
AX = 1A21h (AL='!')
DS:SI -> command string with one or more state characters
CX = length of command string

**Return Registers:**
CF set on error (invalid subfunction)
CF clear if successful

**Details:** The characters in the state string are interpreted left to right, and need not be in any order.

**See also:** Function 1A3Fh

### 8-72 Values of state characters:

| | |
|---|---|
| 'a' | activate driver |
| 'd' | disable driver |
| 'f' | use fast screen output |
| 'g' | always convert gray keys (+ and -) to function keys |
| 'G' | never convert gray keys |
| 'l' | convert gray keys only when ScrollLock active |
| 's' | use slow screen output |
| 't' | Tandy 1000 keyboard (not yet implemented) |

---

### INT 2Fh Function 1A3Ch

*AVATAR.SYS v0.11 (undocumented)*

**UNDETERMINED FUNCTION**

**Registers at call:**
AX = 1A3Ch
*other registers, if any, unknown*

**Return Registers:**
CX = 0000h

---

### INT 2Fh Function 1A3Eh

*AVATAR.SYS v0.11 (undocumented)*

**UNDETERMINED FUNCTION**

**Registers at call:**
AX = 1A3Eh
CL = *unknown*
CH = *unknown*
DL = *unknown*
DH = *unknown*

**Return Registers:**
CL = *unknown*
CH = *unknown*
DL = *unknown*
DH = *unknown*

---

### INT 2Fh Function 1A3Fh

*AVATAR.SYS*

**QUERY DRIVER STATE**

**Registers at call:**
AX = 1A3Fh (AL='?')
ES:DI -> buffer
CX = length of buffer in bytes

**Return Registers:**
CF clear
CX = actual size of returned info

**Details:** The returned information consists of multiple letters whose meanings are described in Table 8-72.

**See also:** Function 1A21h

---

### INT 2Fh Function 1A44h, Subfunction 4156h

*AVATAR.SYS v0.11+*

**GET DATA SEGMENT**

**Registers at call:**
AX = 1A44h
BX = 4156h ('AV')

**Return Registers:**
AX = 0000h
DS = data segment
CX = size of data segment

**Details:** AVATAR.SYS calls this function whenever it is invoked. If each process under a multitasker hooks this function and provides a separate data segment, AVATAR.SYS becomes fully reentrant.

**See also:** Functions 1A21h and 1A3Fh

---

### INT 2Fh Function 1A52h

*AVATAR.SYS v0.11 (undocumented)*

**GET UNKNOWN DATA**

**Registers at call:**
AX = 1A52h
CX = size of buffer
ES:DI -> buffer

**Return Registers:**
*unknown* copied into user buffer

**Details:** The maximum size of the data which may be copied is returned by Function 1A72h.

**See also:** Function 1A72h

---

### INT 2Fh Function 1A53h

*AVATAR.SYS v0.11 (undocumented)*

**UNDETERMINED FUNCTION**

**Registers at call:**
AX = 1A53h
CL = *unknown* (00h-05h)*other registers, if any, unknown*

**Return Registers:**
*unknown*

### INT 2Fh Function 1A72h

*AVATAR.SYS v0.11 (undocumented)*

### *GET UNKNOWN SIZE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1A72h | CX = maximum size of *unknown* |

**See also:** Function 1A52h

### INT 2Fh Function 1A7Bh

*AVATAR.SYS v0.11 (undocumented)*

### *UNDETERMINED FUNCTION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1A7Bh | AX = 0000h<br>CX = 0000h |

### INT 2Fh Function 1A7Dh

*AVATAR.SYS v0.11 (undocumented)*

### *UNDETERMINED FUNCTION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1A7Dh | AX = *unknown* |

### INT 2Fh Function 1AADh

*AVATAR.SYS v0.11 (undocumented)*

### *UNDETERMINED FUNCTION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1AADh<br>DX = 0000h<br>CX = subfunction (00h-0Ch)*other registers, if any, unknown* | AX = 0000h if DX was nonzero<br>*other registers, if any, unknown* |

### INT 49h

*Texas Instruments PC*

### *VIDEO I/O*

**Purpose:** This interrupt apparently provides direct video display on the TI Professional PC.

### INT 59h

*GSS Computer Graphics Interface (GSS*CGI)*

### *INTERFACE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| DS:DX -> block of 5 array pointers | CF set on error<br>AX = error code<br>CF clear if successful<br>AX = return code |

**Details:** INT 59h is the means by which GSS*CGI language bindings communicate with GSS*CGI device drivers and the GSS*CGI device driver controller.

This interface is also used by the IBM Graphic Development Toolkit.

### INT 7Fh

*Halo88*

### *API*

| **Registers at call:** | **Return Registers:** |
|---|---|
| BX = function (see Table 8-73)<br>additional parameters on stack | *unknown* |

**Program:** Halo88 is a suite of graphics routines.

**Details:** According to Stuart Kemp, the code appears to make no provisions for chaining.

**Conflicts:** HDILOAD, Touch Screen, IBM XGA (above), TIGA (Chapter 12), GO32 (Chapter 37), Multilink Advanced (Chapter 49), Btrieve (Chapter 66), Canon IXHND2 (Chapter 66), YTERM (Chapter 66), Alloy 386/Multiware (NI), Convergent CTOS (NI), HLLAPI (NI), Telebit ACS (NI), Novell NetWare (NI)

**8-73 Values for Halo88 function:**

| | |
|---|---|
| 64h | arc |
| 65h | bar |
| 66h | box |
| 67h | circle |
| 68h | clr |
| 69h | default hatch style |
| 6Ah | default line style |
| 6Bh | delhcur |
| 6Ch | delln / deltcur |
| 6Dh | ellipse |
| 6Eh | fill |
| 6Fh | flood |
| 70h | flood2 |
| 71h | init graphics |
| 72h | init hcur |
| 73h | init marker |
| 74h | init tcur |
| 75h | inqarc |
| 76h | inqbknd |
| 77h | inqclr |
| 78h | inqerr |
| 79h | inqgcur |
| 7Ah | inqhcur |
| 7Bh | inqmarker |
| 7Dh | inqtcur |
| 7Eh | inqtext |
| 7Fh | lnabs |
| 80h | lnrel |
| 81h | markerabs |
| 82h | markerrel |
| 83h | moveabs |
| 84h | movehcurabs |
| 85h | movehcurrel |
| 86h | moverel |
| 87h | movetcurabs |
| 88h | movetcurrel |
| 89h | movefrom |
| 8Ah | moveto |
| 8Bh | pie |
| 8Ch | polylnabs |
| 8Dh | polylnrel |
| 8Eh | ptabs |
| 8Fh | ptrel |
| 91h | setasp |
| 92h | set color |
| 93h | set font |
| 94h | set hatch style |
| 95h | set line style |
| 9Ch | setscreen |
| 9Eh | close graphics |
| 9Fh | ftinit |
| A0h | ftlocate |
| A1h | ftext |
| A5h | set viewport |
| A6h | set window |
| A7h | set world |
| AAh | ftcolor |
| ACh | initlp |
| ADh | inqasp |
| AEh | inqdev |
| AFh | inqdisplay |
| B0h | inqft |
| B1h | inqftcolor |
| B2h | inqinterlace |
| B3h | inqlpa |
| B4h | inqlpg |
| B5h | inqmode |
| B6h | inqscreen |
| B7h | inqversion |
| B8h | roam |
| B9h | scroll |
| BAh | setieee |
| BBh | set interlace |
| BCh | shift |
| BDh | start graphics |
| BEh | vpan |
| CBh | gwrite |
| CCh | gread |
| CDh | setxor |
| CEh | rbox |
| CFh | rcir |
| D0h | rlnabs |
| D1h | rlnrel |
| D2h | delbox |
| D3h | delcir |
| D5h | setseg2 |
| DCh | worldoff |
| DDh | mapwtod |
| DEh | mapdtow |
| DFh | mapwton |
| E0h | mapntow |
| E1h | mapdton |
| E2h | mapntod |
| E3h | inqworld |
| E4h | inqviewport |
| E5h | set line width |
| ECh | setstclr |
| EDh | setstang |
| EEh | stext |
| EFh | inqstext |
| F0h | setdegree |
| F1h | inqstsize |
| F2h | polyfabs |
| F3h | polyfrel |
| F4h | inqdrange |
| F5h | inqstang |
| F6h | orglocator |
| F7h | inqlocator |
| F8h | inqarea |
| F9h | setipal |
| FAh | setborder |
| FBh | inqcrange |
| FEh | setclip |
| FFh | fcir |
| 100h | setcrange |
| 101h | setdrange |
| 102h | setlattr |
| 103h | polycabs |
| 104h | polycrel |
| 108h | memcom |
| 109h | memexp |
| 10Ah | memmov |
| 10Eh | movefx |
| 10Fh | movetx |
| 110h | inqrgb |
| 111h | save image |
| 112h | restore image |
| 113h | setapal |
| 114h | setxpal |
| 118h | inqtsize |
| 12Eh | gprint |
| 130h | setprn |
| 131h | setpattr |
| 133h | setbattr |
| 135h | pexpand |
| 136h | ptnorm |
| 137h | pfnorm |
| 13Bh | inqprn |
| 13Ch | lopen |
| 13Dh | lclose |
| 13Eh | lappend |
| 13Fh | lrecord |
| 140h | lswitch |
| 142h | inqfun |

| | | | | | |
|---|---|---|---|---|---|
| 97h | settext | E6h | lnjoint | 15Dh | lsetup |
| 98h | set text color | E7h | set locator | 15Eh | lrest |
| 99h | btext | E8h | read locator | 15Fh | lsave |
| 9Ah | setseg | E9h | setdev | | |
| 9Bh | display | EBh | setstext | | |

---

### INT 7Fh Function 0105h

*HDILOAD.EXE*

### *8514/A VIDEO CONTROLLER INTERFACE*

**Registers at call:**
AX = 0105h

**Return Registers:**
CF set on error
CF clear if successful
CX:DX -> array of FAR pointers to entry points

**Details:** Most functions are invoked by pushing the DWORD parameter block pointer and then performing a FAR call via the appropriate vector of the entry point array.

**Conflicts:** IBM XGA (above), Halo88 (above), Touch Screen (below), TIGA (Chapter 12), GO32 (Chapter 37), Multilink Advanced (Chapter 49), Btrieve (Chapter 66), Canon IXHND2 (Chapter 66), YTERM (Chapter 66), Alloy 386/Multiware (NI), Convergent CTOS (NI), HLLAPI (NI), Telebit ACS (NI), Novell NetWare (NI)

**See also:** Touch Screen Function ABCDh

**8-74 Function numbers: (do FAR call via entry_points+4*function)**

| | | | |
|---|---|---|---|
| 08h | HOPEN | 1Dh | HQMODE |
| 10h | HINT | 22h | HCLOSE |
| 13h | HLDPAL | 30h | HINIT |
| 15h | HBBW | 31h | HSYNC |
| 17h | HBBR | 39h | HSPAL |
| 18h | HBBCHN | 3Ah | HRPAL |

---

### INT 7Fh Function ABCDh

*IBM 8516 Touch Screen Device Driver*

### *GET API ENTRY*

**Registers at call:**
AX = ABCDh
BX = 0000h

**Return Registers:**
AX = total number of functions available
ES:BX -> entry point array

**Conflicts:** Halo88, HDILOAD, IBM XGA (above), TIGA (Chapter 12), GO32 (Chapter 37), Multilink Advanced (Chapter 49), Btrieve (Chapter 66), Canon IXHND2 (Chapter 66), YTERM (Chapter 66), Alloy 386/Multiware (NI), Convergent CTOS (NI), HLLAPI (NI), Telebit ACS (NI), Novell NetWare (NI)

**See also:** HDILOAD Function 0105h

**8-75 Function numbers:**

| | |
|---|---|
| 00h | check initialization and reset |
| 14h | set user-defined subroutine |

**Note**: Each driver function takes two stack parameters using Pascal calling conventions: address of parameter block and address of results buffer
all pointers are FAR pointers
on return,
AX = status
0000h successful
0001h invalid input
0002h interface error
0003h unable to perform function

**8-76 Format of parameter block for function 00h:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | 0000h (function number) |

**Note**: This function should be called before any other device driver functions

**8-77 Format of results buffer for function 00h:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | touch screen status<br>0000h unavailable<br>0001h uncalibrated<br>FFFFh available |
| 02h | WORD | aux mouse status (0000h not present, FFFFh present) |

**Notes**:The following driver parameters will have been reset to zero: touchdown counter, liftoff counter, position at last touch, position at last lift, int call mask, select on count, select off count, pos select on count, pos select off count. The following driver parameters will have been reset as listed:

- mouse emulation mode: left on
- thresholds: 46 on screen, 96 push harder, 80 push release
- x, y hysteresis: 400
- data repeat rate: 40/sec
- select mechanism: push-harder - first-touch
- coordinate origin: upper left corner
- filter frequency: medium
- data block mask: all enabled
- click lock: on

---

**INT F2h** *ICCTSR 1.0*

***ImageCapture COLOR Developer's Kit - API***

**Registers at call:**
AH = function
01h power up
02h power down
03h set controls
04h capture image
05h display image
06h read file
07h write file
08h write array
09h read pixel
0Ah write pixel
0Bh check if VGA present
0Ch set video mode
0Dh check for keystroke
0Eh delay *other registers, if any, unknown*

**Return Registers:**
*unknown*

**Program:** ImageCapture is a product of International Computers
**See also:** ICCTSR INT F3h

---

**INT F3h** *ICCTSR 1.0*

***HANDSHAKE ID VECTOR***

**See also:** ICCTSR INT F2h

## CHAPTER 9

# *SuperVGA*

SuperVGA is informally defined as any video adapter capable of displaying a higher resolution than IBM's VGA (640x480 pixels with 16 simultaneous colors or 320x200 pixels with 256 simultaneous colors) or more than 256 simultaneous colors. This chapter covers the VESA and Sollex SuperVGA BIOS Extensions; the Cirrus Logic, Trident, and Tseng BIOSes; the ET-3000 Zoom TSR; and various miscellaneous SuperVGA-related calls. In addition, many SuperVGA modes are listed in Chapter 8 under the video-mode setting calls.

Although not explicitly mentioned in the remainder of this chapter, all INT 10h calls potentially conflict with the coprocessor error interrupt described in Chapter 3.

## VESA SuperVGA BIOS Extensions

With the proliferation of various Super VGA adapters with incompatible methods for bank-switching memory and performing other operations required to access resolutions beyond those of IBM's VGA came the problem of having software support a wide variety of different adapters. The Video Electronics Standards Association, to which virtually all video adapter manufacturers now belong, created its SuperVGA BIOS Extensions (SBE) to address this problem and provide a standard interface which would allow graphics code to be programmed once and yet run on virtually all SuperVGA adapters ever produced. The SuperVGA BIOS Extensions may be integrated into the adapter's video ROM or be provided as a small (typically 2-5K) TSR.

---

### INT 10h Function 4F00h

*VESA SuperVGA BIOS*

### *GET SuperVGA INFORMATION*

**Purpose:** Determine whether VESA BIOS extensions are present and the capabilities supported by the display adapter.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4F00h | AL = 4Fh function supported |
| ES:DI -> 256-byte buffer for SuperVGA information (see Table 9-1) | AH = status<br>00h successful<br>ES:DI buffer filled<br>01h failed |

**See also:** Function 4F01h, SOLLEX Function 7F00h (below)

**9-1 Format of SuperVGA information:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 4 BYTEs | signature ('VESA') |
| 04h | WORD | VESA version number |
| 06h | DWORD | pointer to OEM name<br>"761295520" for ATI |

| | | |
|---|---|---|
| 0Ah | 4 BYTEs | capabilities |
| 0Eh | DWORD | pointer to list of supported VESA and OEM video modes (list of words terminated with FFFFh) |
| 12h | 238 BYTEs | reserved |

**Note:** The list of supported video modes is stored in the reserved portion of the SuperVGA information record by some implementations, and it may thus be necessary to either copy the mode list or use a different buffer for all subsequent VESA calls.

### INT 10h Function 4F01h

*VESA SuperVGA BIOS*

### *GET SuperVGA MODE INFORMATION*

**Purpose:** Determine the attributes of the specified video mode.

**Registers at call:**

AX = 4F01h
CX = SuperVGA video mode
ES:DI -> 256-byte buffer for mode information (see Table 9-2)

**Return Registers:**

AL = 4Fh function supported
AH = status
  00h successful
    ES:DI buffer filled
  01h failed

**See also:** Functions 4F00h and 4F02h

**9-2 Format of mode information:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | mode attributes<br>bit 0: mode supported<br>bit 1: optional information available<br>bit 2: BIOS output supported<br>bit 3: set if color, clear if monochrome<br>bit 4: set if graphics mode, clear if text mode |
| 02h | BYTE | window A attributes<br>bit 0: exists<br>bit 1: readable<br>bit 2: writable<br>bits 3-7 reserved |
| 03h | BYTE | window B attributes (as for window A) |
| 04h | WORD | window granularity in KB |
| 06h | WORD | window size in KB |
| 08h | WORD | start segment of window A |
| 0Ah | WORD | start segment of window B |
| 0Ch | DWORD | -> FAR window positioning function (equivalent to Function 4F05h) |
| 10h | WORD | bytes per scan line |
| **---remainder is optional for VESA modes in v1.0/1.1, needed for OEM modes---** | | |
| 12h | WORD | width in pixels |
| 14h | WORD | height in pixels |
| 16h | BYTE | width of character cell in pixels |
| 17h | BYTE | height of character cell in pixels |
| 18h | BYTE | number of memory planes |
| 19h | BYTE | number of bits per pixel |
| 1Ah | BYTE | number of banks |
| 1Bh | BYTE | memory model type (see Table 9-3) |
| 1Ch | BYTE | size of bank in KB |
| 1Dh | BYTE | number of image pages |
| 1Eh | BYTE | reserved (0) |

| | | |
|---|---|---|
| **---VBE v1.2+---** | | |
| 1Fh | BYTE | red mask size |
| 20h | BYTE | red field position |
| 21h | BYTE | green mask size |
| 22h | BYTE | green field size |
| 23h | BYTE | blue mask size |
| 24h | BYTE | blue field size |
| 25h | BYTE | reserved mask size |
| 26h | BYTE | reserved mask position |
| 27h | BYTE | direct color mode info |
| 28h | 216 BYTEs | reserved (0) |

**9-3 Values for memory model type:**

| | |
|---|---|
| 00h | text |
| 01h | CGA graphics |
| 02h | HGC graphics |
| 03h | 16-color (EGA) graphics |
| 04h | packed pixel graphics |
| 05h | "sequ 256" (non-chain 4) graphics |
| 06h | direct color (HiColor, 24-bit color) |
| 07h | YUV (luminance-chrominance, also called YIQ) |
| 08h-0Fh | reserved for VESA |
| 10h-FFh | OEM memory models |

## INT 10h Function 4F02h

*VESA SuperVGA BIOS*

### *SET SuperVGA VIDEO MODE*

**Registers at call:**

AX = 4F02h

BX = video mode (see Table 9-4; bit 15 set means don't clear video memory)

**Return Registers:**

AL = 4Fh function supported

AH = status

00h successful

01h failed

**See also:** Functions 4F01h and 4F03h

**9-4 Values for VESA video mode:**

| | |
|---|---|
| 00h-FFh | OEM video modes (see Function 00h) |
| 100h | 640x400x256 |
| 101h | 640x480x256 |
| 102h | 800x600x16 |
| 103h | 800x600x256 |
| 104h | 1024x768x16 |
| 105h | 1024x768x256 |
| 106h | 1280x1024x16 |
| 107h | 1280x1024x256 |
| 108h | 80x60 text |
| 109h | 132x25 text |
| 10Ah | 132x43 text |
| 10Bh | 132x50 text |
| 10Ch | 132x60 text |
| **---VBE v1.2---** | |
| 10Dh | 320x200x32K |
| 10Eh | 320x200x64K |

| | |
|---|---|
| 10Fh | 320x200x16M |
| 110h | 640x480x32K |
| 111h | 640x480x64K |
| 112h | 640x480x16M |
| 113h | 800x600x32K |
| 114h | 800x600x64K |
| 115h | 800x600x16M |
| 116h | 1024x768x32K |
| 117h | 1024x768x64K |
| 118h | 1024x768x16M |
| 119h | 1280x1024x32K |
| 11Ah | 1280x1024x64K |
| 11Bh | 1280x1024x16M |

**See also:** Tables 8-2, 8-19, 8-27, and 8-37

**9-5 Values for S3 OEM video mode:**

| | |
|---|---|
| 201h | 640x480x256 |
| 202h | 800x600x16 |
| 203h | 800x600x256 |
| 204h | 1024x768x16 |
| 205h | 1024x768x256 |
| 206h | 1280x960x16 |
| 208h | 1280x1024x16 |
| 301h | 640x480x32K |

## INT 10h Function 4F03h

*VESA SuperVGA BIOS*

### *GET CURRENT VIDEO MODE*

**Registers at call:**
AX = 4F03h

**Return Registers:**
AL = 4Fh function supported
AH = status
    00h successful
    01h failed
BX = video mode (see Tables 9-4 and 9-5)

**See also:** Function 4F02h

## INT 10h Function 4F04h

*VESA SuperVGA BIOS*

### *SAVE/RESTORE SuperVGA VIDEO STATE*

**Registers at call:**
AX = 4F04h
CX = flags for states to save/restore
    bit 0: video hardware state
    bit 1: video BIOS data state
    bit 2: video DAC state
    bit 3: SuperVGA state
DL = subfunction
    00h get state buffer size
    01h save video states
        ES:BX -> buffer
    02h restore video states
        ES:BX -> buffer

**Return Registers:**
AL = 4Fh function supported
AH = status
    00h successful
    01h failed

BX = number of 64-byte blocks needed

---

## INT 10h Function 4F05h

*VESA SuperVGA BIOS*

### *CPU VIDEO MEMORY CONTROL*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4F05h | AL = 4Fh function supported |
| BH = subfunction | AH = status |
| 00h select video memory window | 00h successful, 01h failed |
| DX = window address in video memory (in granularity units) | |
| 01h get video memory window | DX = window address in video memory (in granularity units) |
| | BL = window number |
| | 00h window A |
| | 01h window B |

**See also:** Functions 4F01h, 4F06h, and 4F07h; Everex Function 7000h Subfunction 0004h (Chapter 8)

---

## INT 10h Function 4F06h

*VESA SuperVGA BIOS v1.1+*

### *GET/SET LOGICAL SCAN LINE LENGTH*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4F06h | AL = 4Fh if function supported |
| BL = function | AH = status |
| | 00h successful, 01h failed |
| 00h set scan line length | |
| CX = desired width in pixels | |
| 01h get scan line length | BX = bytes per scan line |
| | CX = number of pixels per scan line |
| | DX = maximum number of scan lines |

**Details:** If the desired width is not achievable, the next larger width will be set.

The scan line may be wider than the visible area of the screen.

This function is valid in text modes, provided that values are multiplied by the character cell width/height.

**See also:** Functions 4F01h, 4F05h, and 4F07h

---

## INT 10h Function 4F07h, Subfunction 00h

*VESA SuperVGA BIOS v1.1+*

### *GET/SET DISPLAY START*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4F07h | AL = 4Fh if function supported |
| BH = 00h (reserved) | AH = status |
| BL = function | 00h successful, 01h failed |
| 00h set display start | |
| CX = leftmost displayed pixel in scan line | |
| DX = first displayed scan line | |
| 01h get display start | BH = 00h |
| | CX = leftmost displayed pixel in scan line |
| | DX = first displayed scan line |

**Details:** This function is valid in text modes, provided that values are multiplied by the character cell width/height.

**See also:** Functions 4F01h, 4F05h, and 4F06h

---

### INT 10h Function 4F08h

*VESA SuperVGA BIOS v1.2+*

### *GET/SET DAC PALETTE CONTROL*

**Registers at call:**
AX = 4F08h
BL = function
  00h set DAC palette width
    BH = desired number of bits per primary color
  01h get DAC palette width

**Return Registers:**
AL = 4Fh if function supported
AH = status

BH = current number of bits per primary color
  (06h = standard VGA)

---

### INT 10h Function 4FFFh

*VESA SuperVGA BIOS*

### *Everex - TURN VESA ON/OFF*

**Registers at call:**
AX = 4FFFh
DL = new state (00h off, 01h on)

**Return Registers:**
AX = 0000h if successful

## SOLLEX SuperVGA Extensions

The SOLLEX (Smos videO controLLer EXtensions) SuperVGA functions are an extension to the VESA SuperVGA BIOS Extensions (see AX=4F00h) by Seiko Epson Corporation intended to provided a standardized interface to SuperVGA functionality not addressed by the VESA standard.

---

### INT 10h Function 7F00h

*SOLLEX SuperVGA*

### *GET EXTENSIONS INFO*

**Registers at call:**
AX = 7F00h

**Return Registers:**
AL Ö 7Fh if SOLLEX extensions not supported
AL = 7Fh if supported
  AH = status
    00h successful
      ES:DI -> info structure (see Table 9-6)
    01h failed

**See also:** Function 4F00h

**9-6 Format of info structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to VESA function dispatch table |
| 04h | DWORD | pointer to SOLLEX function dispatch table |
| 08h | DWORD | pointer to VESA SuperVGA info (see Table 9-1) |
| 0Ch | DWORD | pointer to mode info structure table, consisting of alternating ResInfo (see Table 9-7) and VESA mode information (see Table 9-2) blocks, terminated with an FFFFh word |
| 10h | DWORD | pointer to font info structure table (see Table 9-8) |
| 14h | WORD | high resolution crystal frequency in Hz (0000h = not present) |
| 16h | DWORD | pointer to ASCIZ ID string |
| 1Ah | DWORD | pointer to timeout reset table<br>array of bytes, each a multiple of the minimum time increment |
| 1Eh | WORD | minimum time increment in timer ticks |
| 20h | BYTE | inverse options supported<br>bit 0: inverse supported<br>bits 1-7: reserved |
| 21h | BYTE | normal color value |

| | | |
|---|---|---|
| 22h | BYTE | inverse color value |
| 23h | WORD | port to be accessed for normal/inverse settings |
| 25h | WORD | type of interface chip (currently undefined) |
| 27h | WORD | program operational mode<br>bits 1-0: 00 no preference<br>01 terse (minimum detail in program messages)<br>10 verbose<br>11 use menus if supported, verbose mode otherwise<br>bits 7-2: reserved |
| 29h | WORD | SOLLEX specification version |
| 2Bh | WORD | version of VESA/SOLLEX implementation |
| 2Dh | DWORD | offset to relocatable portion of SOLLEX extensions (for CONFIG) |
| 2Eh | DWORD | offset to unused section of the extensions ROM |
| 31h | 16 BYTEs | reserved |

**Note:** All DWORD pointers initially require segment fixups; if the segment is 0000h, it should be changed to the returned ES, otherwise it may be assumed to be correct.

**9-7 Format of ResInfo:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | 16-bit mode number |
| 02h | WORD | adapter type (00h VGA, 01h EGA, 02h CGA, 03h MDA) |
| 04h | WORD | display info (see Table 9-11) |
| 06h | DWORD | pointer to video parameter table |
| 0Ah | BYTE | replacement entry in master Video Parameter |
| 0Bh | BYTE | mode requested for mode set by BIOS |
| 0Ch | DWORD | pointer to LoadReg table |
| 10h | BYTE | index into table of clock values (see Table 9-9) |

**9-8 Format of font info table [array] entry:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | required font height |
| 01h | BYTE | parameter to load text mode font |
| 02h | BYTE | parameter to load graphics mode font |

**9-9 Values for clock value index:**

| | | | |
|---|---|---|---|
| 00h | 25 MHz | 08h | 25 MHz |
| 01h | 28 MHz | 09h | 28 MHz |
| 02h | PCLK | 0Ah | 36 MHz |
| 03h | 31.5 MHz (VESA 640x480) | 0Bh | 45 MHz (for 1024x768) |
| 04h | reserved | 0Ch | 80 MHz |
| 05h | 16 MHz (EGA) | 0Dh | 40 MHz |
| 06h | PCLK | 0Eh | 65 MHz |
| 07h | 24 MHz (EGA) | 0Fh | 1 MHz (for powerdown) |

## INT 10h Function 7F01h, Subfunction 00h

*SOLLEX SuperVGA*

### *ADAPTER CONTROL - SET ADAPTER*

**Registers at call:**
AX = 7F01h
BL = 00h
CX = adapter request (see Table 9-10)

**Return Registers:**
AL = 7Fh if supported
AH = status
(00h successful, 01h failed)

**Details:** Initializes video hardware to a particular standard.

**See also:** Function 7F00h, Function 7F01h Subfunctions 01h and 02h

### 9-10 Bitfields for adapter request

| | |
|---|---|
| bits 1-0 | adapter type (00 VGA, 01 EGA, 10 CGA, 11 MDA) |
| bit 2 | reserved |
| bits 4-3 | change displays (00 none, 01 analog 10 digital 11 panel) |
| bits 6-5 | desired monitor sense (01 color, 10 mono, 11=8514) |
| bit 7 | lock override |
| bit 8 | alternate adapter mode |
| bits 15-9 | reserved |

---

## INT 10h Function 7F01h, Subfunction 01h

*SOLLEX SuperVGA*

### *ADAPTER CONTROL - GET ADAPTER*

**Registers at call:**
AX = 7F01h
BL = 01h

**Return Registers:**
AL = 7Fh if supported
AH = status
00h successful, 01h failed
BX = adapter type (see Table 9-10)
DX = display type (see Table 9-11)

**See also:** Function 7F00h, Function 7F01h Subfunctions 00h and 02h

### 9-11 Bitfields for display type:

| | |
|---|---|
| bits 1-0 | monitor sense (00 none, 01 color, 10 mono, 11=8514) |
| bit 2 | multi-frequency analog monitor active |
| bit 3 | LCD panel active |
| bit 4 | plasma/electroluminescent panel active |
| bit 5 | PS/2-type monitor active |
| bit 6 | multi-frequency digital monitor active |
| bit 7 | Enhanced Color Display monitor active |
| bit 8 | alternate display active |
| bits 15-9 | reserved |

---

## INT 10h Function 7F01h, Subfunction 02h

*SOLLEX SuperVGA*

### *ADAPTER CONTROL - DETERMINE ADAPTER SUPPORT*

**Registers at call:**
AX = 7F01h
BL = 02h
CX = adapter request (see Function 7F01h Subfunction 00h)

**Return Registers:**
AL = 7Fh if supported
AH = status
00h requested setting can successfully be made
01h requested setting not available in this configuration

**See also:** Function 7F00h, Function 7F01h Subfunction 00h

---

## INT 10h Function 7F02h, Subfunction 00h

*SOLLEX SuperVGA*

### *DISPLAY OUTPUT CONTROL - SET DISPLAY OUTPUT*

**Registers at call:**
AX = 7F02h
BL = 00h
CX = display output setting (see Table 9-12)

**Return Registers:**
AL = 7Fh if supported
AH = status
00h successful, 01h failed

**See also:** Function 7F00h, Function 7F02h Subfunction 00h

### 9-12 Bitfields for display output setting:

| | |
|---|---|
| bit 0 | CRTC control in bits 2,1 valid |
| bit 1 | enable digital output |
| bit 2 | enable analog output |
| bit 3 | panel control in bits 5,4 valid |

| | |
|---|---|
| bit 4 | enable LCD output |
| bit 5 | enable plasma/EL output |
| bit 6 | inverse control in bit 7 valid |
| bit 7 | 0=normal, 1=inverse |
| bits 15-8 | reserved |

---

### INT 10h Function 7F02h, Subfunction 01h
*SOLLEX SuperVGA*
### *DISPLAY OUTPUT CONTROL - GET DISPLAY OUTPUT*

**Registers at call:**
AX = 7F02h
BL = 01h

**Return Registers:**
AL = 7Fh if supported
  AH = status
    00h successful
      BX = display output setting (see Table 9-12)
      CX = displays attached (see Table 9-13)
    01h failed

**See also:** Function 7F00h, Function 7F02h Subfunction 00h

**9-13 Bitfields for displays attached:**

| | |
|---|---|
| bit 0 | PS/2 display on analog output |
| bit 1 | multi-frequency monitor on analog output |
| bit 2 | LCD panel attached |
| bit 3 | plasma/electroluminescent panel attached |
| bit 4 | multi-frequency monitor on digital output |
| bit 5 | Enhanced Color Display attached to digital outpt |
| bit 6 | alternate display |
| bits 15-7 | reserved |

---

### INT 10h Function 7F03h, Subfunction 00h
*SOLLEX SuperVGA*
### *VIDEO SUPPORT CONTROL - GET SUPPORT INFO*

**Registers at call:**
AX = 7F03h
BL = 00h
CX = support type (see Table 9-14)

**Return Registers:**
AL = 7Fh if supported
  AH = status
    00h successful
      CX = size of video support code
      DX = segment of physical video support (0000h if no ROM)
      ES = segment of active video support
      ES:DI -> information block (DI = 0000h if none available)
    01h failed

**See also:** Function 7F03h Subfunction 01h

**9-14 Values for support type:**

| | |
|---|---|
| 0000h | VGA |
| 0001h | EGA |
| 0002h | CGA |
| 0003h | MDA |
| 0004h | extensions |
| 0005h-0012h | reserved for SOLLEX |
| 0013h | Hercules |
| 0014h-001Fh | reserved for SOLLEX |
| 0020h-00FFh | reserved for OEM |

### INT 10h Function 7F03h, Subfunction 01h

*SOLLEX SuperVGA*

### *VIDEO SUPPORT CONTROL - INITIALIZE VIDEO SUPPORT*

**Registers at call:**
AX = 7F03h
BL = 01h
CX = support request (see Table 9-14)
ES = segment of support code

**Return Registers:**
AL = 7Fh if supported
AH = status
00h successful
01h failed

**Details:** Initializes the indicated video support by calling ES:0003h; this function may be used to switch the active video support back to ROM after Function 7F03h Subfunction 02h.

**See also:** Function 7F03h Subfunction 00h

### INT 10h Function 7F03h, Subfunction 02h

*SOLLEX SuperVGA*

### *VIDEO SUPPORT CONTROL - GO RAM RESIDENT*

**Registers at call:**
AX = 7F03h
BL = 02h
CX = support request (see Table 9-14)
ES = destination segment

**Return Registers:**
AL = 7Fh if supported
AH = status
00h successful
01h failed

**See also:** Function 7F00h, Function 7F03h Subfunction 01h

### INT 10h Function 7F04h, Subfunction 00h

*SOLLEX SuperVGA*

### *POWER CONTROL - SET POWER STATE*

**Registers at call:**
AX = 7F04h
BL = 00h
CX = new power state

**Return Registers:**
AL = 7Fh if supported
AH = status
00h successful, 01h failed

**Details:** Higher values progressively reduce the operations available on the video adapter while yielding increasing power savings.

**See also:** Function 7F00h, Function 7F04h Subfunction 01h

### INT 10h Function 7F04h, Subfunction 01h

*SOLLEX SuperVGA*

### *POWER CONTROL - GET POWER STATE*

**Registers at call:**
AX = 7F04h
BL = 01h

**Return Registers:**
AL = 7Fh if supported
AH = status
00h successful
CX = current power state
DX = maximum state
01h failed

**See also:** Function 7F00h, Function 7F04h Subfunction 00h

### INT 10h Function 7F04h, Subfunction 02h

*SOLLEX SuperVGA*

### *POWER CONTROL - SET TIMEOUT RESET*

**Registers at call:**
AX = 7F04h
BL = 02h
CX = timeout reset

**Return Registers:**
AL = 7Fh if supported
AH = status
00h successful, 01h failed

**See also:** Function 7F00h, Function 7F04h Subfunction 03h

## INT 10h Function 7F04h, Subfunction 03h

*SOLLEX SuperVGA*

### *POWER CONTROL - GET TIMEOUT RESET*

**Registers at call:**
AX = 7F04h
BL = 03h

**Return Registers:**
AL = 7Fh if supported
  AH = status
    00h successful
      BX = time increment
      CX = current timeout reset
      DX = maximum timeout reset
      ES:DI -> timeout reset table (array of bytes)
    01h failed

**Details:** The timeout period is computed as (BYTE ES:[DI+CX]) * BX timer ticks.
**See also:** Function 7F00h, Function 7F04h Subfunction 02h

## INT 10h Function 7F05h

*SOLLEX SuperVGA*

### *LOAD REGISTER*

**Registers at call:**
AX = 7F05h
ES:DI -> register value table (see Table 9-15)

**Return Registers:**
AL = 7Fh if supported
  AH = status
    00h successful, 01h failed

**See also:** EGA RIL Functions F1h, F3h, and F5h (Chapter 8)

**9-15 Format of register value table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | base I/O register (FFFFh = end of list) |
| 02h | 2N BYTEs | pairs of values to be written to the base I/O register as an index value and the following register as a data byte |
| 2N+2 | WORD | FFFFh (end of data list) |
| | | ... (repeats until FFFFh base address) |

## INT 10h Function 7F06h

*SOLLEX SuperVGA*

### *MULTIPLE FONT CONTROL*

**Registers at call:**
AX = 7F06h
BL = subfunction

**Return Registers:**
AL = 7Fh if supported
  AH = status
    00h successful, 01h failed

  00h set multiple font state
    CX = new state (00h off, 01h on)
  01h get multiple font state — BL = current state (00h off, 01h on)

**See also:** Standard Function 11h (Chapter 8)

## INT 10h Function 7F07h

*SOLLEX SuperVGA*

### *FILL VIDEO RAM*

**Registers at call:**
AX = 7F07h
BL = how much to fill
  00h regen size
  01h all video memory
CX = pattern to write (normally 0720h
  for text modes and 0000h for graphics)

**Return Registers:**
AL = 7Fh if supported
  AH = status
    00h successful
    01h failed

**See also:** Standard Function 06h (Chapter 8)

## Cirrus Logic

---

### INT 10h Function 12h, Subfunction 80h

*Cirrus Logic BIOS*

***INQUIRE VGA TYPE***

**Registers at call:**
AH = 12h
BL = 80h

**Return Registers:**
AX = controller type (see Table 9-16)
BL = silicon revision number (bit 7 set if not available)

**See also:** Function 12h Subfunctions 81h, 82h, 85h, 9Ah, and A1h

**9-16 Values for controller type:**

| | |
|---|---|
| 0000h | no extended alternate select support |
| 0002h | 510/520 |
| 0003h | 610/620 |
| 0004h | 5320 |
| 0005h | 6410 |
| 0006h | 5410 |
| 0007h | 6420 |
| 0008h | 6412 |
| 0010h | 5401 |
| 0011h | 5402 |
| 0012h | 5420 |
| 0013h | 5422 |
| 0014h | 5424 |
| 0015h | 5426 |
| 0016h | 5420r1 |
| 0017h | 5402r1 |
| 0018h | 5428 |
| 0020h | 6205/15/25 |
| 0030h | 5432 |
| 0031h | 5434 |

---

### INT 10h Function 12h, Subfunction 81h

*Cirrus Logic BIOS*

***GET BIOS VERSION NUMBER***

**Registers at call:**
AH = 12h
BL = 81h

**Return Registers:**
AH = BIOS major version
AL = BIOS minor version

**See also:** Function 12h Subfunctions 80h and 82h

---

### INT 10h Function 12h, Subfunction 82h

*Cirrus Logic BIOS*

***GET DESIGN REVISION CODE***

**Registers at call:**
AH = 12h
BL = 82h

**Return Registers:**
AL = chip revision

**See also:** Function 12h Subfunctions 80h and 81h

---

### INT 10h Function 12h, Subfunction 85h

*Cirrus Logic BIOS*

***GET INSTALLED MEMORY***

**Registers at call:**
AH = 12h
BL = 85h

**Return Registers:**
AL = number of 64K banks of video memory

**See also:** Function 12h Subfunctions 80h and 93h

### INT 10h Function 12h, Subfunction 93h

*Cirrus Logic BIOS*

### *FORCE 8-BIT OR 16-BIT OPERATION*

**Registers at call:**
AH = 12h
BL = 93h
AL = new I/O width (00h = 16 bits, 01h = 8 bits)

**Return Registers:**
none

**See also:** Function 12h Subfunctions 9Ah

---

### INT 10h Function 12h, Subfunction 9Ah

*Cirrus Logic BIOS*

### *GET USER OPTIONS*

**Registers at call:**
AH = 12h
BL = 9Ah

**Return Registers:**
AX = options word 1 (see Table 9-17)
CX = options word 2 (see Table 9-18)

**See also:** Function 12h Subfunctions 80h, 93h, A0h, A3h, and A4h

**9-17 Bitfields for options word 1:**

| | |
|---|---|
| bits 0,1 | reserved |
| bits 2-4 | monitor type |
| bits 5-6 | maximum vertical resolution |
| bits 7-9 | reserved |
| bit 10 | force 8-bit operation |
| bits 11-13 | reserved |
| bit 14 | vertical refresh frequency at 640x480 |
| bit 15 | reserved |

**9-18 Bitfields for options word 2:**

| | |
|---|---|
| bits 0-3 | reserved |
| bits 4-5 | vertical refresh frequency at 1280x1024 |
| bits 6-10 | reserved |
| bits 11-12 | vertical refresh frequency at 800x600 |
| bits 13-15 | vertical refresh frequency at 1024x768 |

---

### INT 10h Function 12h, Subfunction A0h

*Cirrus Logic BIOS*

### *GET VIDEO MODE AVAILABILITY*

**Registers at call:**
AH = 12h
BL = A0h
AL = video mode number (00h-7Fh)

**Return Registers:**
AH bit 0: video mode supported
BX = offset of BIOS subroutine to fixup standard video parameters (call subroutine with DS:SI and ES:DI as returned by this call)
DS:SI -> standard video parameters or FFFFh:FFFFh
ES:DI -> supplemental video parameters or FFFFh:FFFFh

**See also:** Function 12h Subfunctions 9Ah and A1h, BIOS Function 00h (Chapter 8)

---

### INT 10h Function 12h, Subfunction A1h

*Cirrus Logic BIOS*

### *READ MONITOR TYPE AND ID FROM 15-PIN CONNECTOR*

**Registers at call:**
AH = 12h
BL = A1h

**Return Registers:**
BH = monitor ID (see Table 9-19)
BL = monitor type (00h color, 01h grayscale, 02h no display)

**See also:** Function 12h Subfunctions 9Ah and A2h

**9-19 Values for monitor ID:**

| | |
|---|---|
| 00h-08h | reserved |
| 09h | IBM 8604/8507 or equivalent |
| 0Ah | IBM 8514 or equivalent |
| 0Bh | IBM 8515 or equivalent |
| 0Dh | IBM 8503 or equivalent |
| 0Eh | IBM 8512/8513 or equivalent |
| 0Fh | no monitor |

---

### INT 10h Function 12h, Subfunction A2h
### *SET MONITOR HORIZONTAL RETRACE FREQUENCY*

*Cirrus Logic BIOS*

**Registers at call:**
AH = 12h
BL = A2h
AL = retrace rate (see Table 9-20)

**Return Registers:**
none

**See also:** Function 12h Subfunctions 9Ah, A0h, and A4h

**9-20 Values for retrace rate:**

| | |
|---|---|
| 00h | standard VGA (31.5 kHz) |
| 01h | 8514-compatible (31.5 kHz and 35.5 kHz interlaced) |
| 02h | SuperVGA (31.5-35.1 kHz) |
| 03h | extended SuperVGA (31.5-35.5 kHz) |
| 04h | multi-frequency (31.5-37.8 kHz) |
| 05h | extended multi-frequency (31.5-48.0 kHz) |
| 06h | super multi-frequency (31.5-56.0 kHz) |
| 07h | extended super multi-frequency (31.5-64.0 kHz) |

---

### INT 10h Function 12h, Subfunction A2h
### *SET VGA REFRESH*

*Cirrus Logic BIOS*

**Registers at call:**
AH = 12h
BL = A3h
AL = refresh rate for 640x480 (00h normal, 01h high)

**Return Registers:**
none

**See also:** Function 12h Subfunctions 9Ah and A4h

---

### INT 10h Function 12h, Subfunction A4h
### *SET MONITOR TYPE*

*Cirrus Logic BIOS*

**Registers at call:**
AH = 12h
BL = A4h
AL bits 3-0 = maximum vertical resolution
    (0 = 480 scanlines, 1 = 600, 2 = 768, 3 = 1024, other reserved)
AL bits 7-4 = vertical refresh at 640x480
    (0 = 60Hz, 1 = 72Hz, other reserved)
BH bits 3-0 = vertical refresh at 800x600
    (0 = 56Kz, 1 = 60Hz, 2 = 72Hz, other reserved)
BH bits 7-4 = vertical refresh at 1024x768
    (0=87Hz-int, 1=60Hz, 2=70Hz, 3=72Hz, 4=76Hz, other reserved)
CH bits 4-7 = vertical refresh at 1280x1024
    (0=87Hz-int, 1=60Hz, 2=70Hz, other reserved)

**Return Registers:**
none

**See also:** Function 12h Subfunctions 9Ah and A3h

## Trident Video BIOS

### INT 10h Function 12h, Subfunction 11h

*Trident TVGA 8800+ BIOS*

***GET BIOS INFO***

**Registers at call:**
AH = 12h
BL = 11h

**Return Registers:**
AL = 12h if function supported
ES:BP -> BIOS info structure (see Table 9-21)

**See also:** Function 12h Subfunction 12h

**9-21 Format of Info structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | *unknown* (=0) |
| 01h | BYTE | OEM Code (00h for original Trident) |
| 02h | WORD | *ID* (1073h for 8800BR, 1074 for 8800CS) |
| 04h | 8 BYTEs | BIOS date ('mm/dd/yy') |
| 0Ch | WORD | *unknown* |
| 0Eh | 8 BYTEs | BIOS Version (' C3-128 ', ' C3-129 ') |

### INT 10h Function 12h, Subfunction 12h

*Trident TVGA 8800+ BIOS*

***GET VIDEO RAM SIZE***

**Registers at call:**
AH = 12h
BL = 12h

**Return Registers:**
AL = 12h if function supported
AH = number of 256K banks of RAM installed

**See also:** Function 12h Subfunction 11h

### INT 6Dh

*Trident SVGA*

***VIDEO BIOS HANDLER***

**Conflicts:** VGA INT 6Dh (Chapter 8)

**Details:** The BIOS INT 10h handler on various Trident VGA cards consists merely of a call to INT 6Dh followed by an IRET.

## Tseng Video BIOS

The Tseng HiColor BIOS extensions are supported by: Diamond Computer Systems SpeedStar HiColor VGA, Everex Systems HC VGA, Focus Information Systems 2theMax 4000, Cardinal Technologies VGA732, Orchid ProDesigner IIs, Genoa 7900, Boca Research Super X VGA, and others.

### INT 10h Function 10F0h

*Tseng ET-4000 BIOS*

***SET HiColor GRAPHICS MODE***

**Purpose:** Switch the display into a graphics mode with 15 or more bits per pixel.

**Registers at call:**
AX = 10F0h
BL = video mode (see also Tables 8-2, 8-27, and 9-4)
    13h = 320x200x32K
    2Dh = 640x350x32K
    2Eh = 640x480x32K
    2Fh = 640x400x32K
    30h = 800x600x32K
    3Eh = 640x480x16M (Genoa 7900)
    FFh Tseng 24-bit color mode

BH = mode
2Eh = 640x480x16M

**Return Registers:**
AL = 10h if supported
AH = status
00h if successful
other on error

**See also:** Functions 10F1h and 10F2h, Diamond Speedstar Function 10E0h (below), standard Function 00h (Chapter 8)

---

### INT 10h Function 10F1h
*Tseng ET-4000 BIOS*

### *GET DAC TYPE*

**Purpose:** Determine which type of digital-to-analog converter is installed on the display board.

**Registers at call:**
AX = 10F1h

**Return Registers:**
AL = 10h if supported
BL = type of digital/analog converter
00h normal VGA DAC
01h Sierra SC1148x HiColor DAC
02h new Sierra DAC
else other HiColor DAC

**See also:** Functions 10F0h and 10F2h

---

### INT 10h Function 10F2h
*Tseng ET-4000 BIOS*
*(partially documented)*

### *CHECK IF IN HiColor MODE*

**Purpose:** Determine whether the display is in a graphics mode with 15 or more bits per pixel color resolution.

**Registers at call:**
AX = 10F2h
BL = 00h

**Return Registers:**
AX = 0010h if supported
BL = video mode type
00h normal
else HiColor mode

**See also:** Functions 10F0h and 10F1h, standard Function 0Fh (Chapter 8)

---

### INT 10h Function 12h, various subfuncs
*Tseng ET-4000 BIOS v3.00*

### *BUG*

**Registers at call:**
AH = 12h
BL = 2Eh, 74h, 80h, BEh, C6h, and F0h

**Return Registers:**
crash

**Details:** Due to an omitted end-of-list marker, this version of the BIOS will crash the system on these functions.

---

### INT 10h Function 1Dh
*Tseng ET-4000 BIOS v3.00*

### *BUG*

**Registers at call:**
AH = 1Dh

**Return Registers:**
crash

**Details:** This version of the BIOS jumps to a random location on this function due to a fencepost error.

## OPTIMA/ET-3000 Zoom TSR

The Tseng ET-3000 chipset contains hardware zooming features, which may be accessed through a special TSR. Not all vendors of ET-3000-based display adapters include the Tseng TSR which supports these functions.

The default interrupt for the zooming TSR's API is INT 61h, but it may be reconfigured. The actual interrupt number in use can be obtained by calling INT 16h Function FFh.

### INT 10h Function 10F1h

---

### INT 16h Function FFh
*OPTIMA 1024 VGA-Sync,ET-3000 chipset*

### *QUERY ZOOM INTERRUPT*

**Registers at call:**
AH = FFh

**Return Registers:**
AL = interrupt number to which BIOS keyboard handler has been relocated
AL+1 = Zoom interrupt number
BX = hotkey

**Details:** The default interrupts are 60h for keyboard and 61h for Zoom interrupt; the default hot key is F10.

Not all vendors include the Tseng TSR which supports these functions.

**Conflicts:** KBUF (Chapter 13), PC Tools (Chapter 61)

---

### INT 61h Function 0000h
*OPTIMA 1024 VGA-Sync,ET-3000 chipset*

### *ZOOM DISPLAY*

**Purpose:** Zoom the display based on the specified zoom factor.

**Registers at call:**
AX = 0000h
BX = zoom factor (0-7)

**Return Registers:**
none

**Conflicts:** see Chapter 2

**See also:** Functions 0001h, 0002h, and 0005h; INT 16h Function FFh

---

### INT 61h Function 0001h
*OPTIMA 1024 VGA-Sync,ET-3000 chipset*

### *CENTER ZOOM WINDOW*

**Purpose:** Positions the zoom window such that the specified window-relative coordinates appear as close as possible to the center of the display. Useful for scrolling and panning.

**Registers at call:**
AX = 0001h
BX = X coordinate to center
CX = Y coordinate to center

**Return Registers:**
none

**Conflicts:** see Chapter 2

**See also:** Functions 0000h, 0002h, and 0005h; INT 16h Function FFh

---

### INT 61h Function 0002h
*OPTIMA 1024 VGA-Sync,ET-3000 chipset*

### *END ZOOM*

**Purpose:** Switches off zoom and returns window to its original state.

**Registers at call:**
AX = 0002h

**Return Registers:**
none

**Conflicts:** see Chapter 2

**See also:** Function 0000h, INT 16h Function FFh

---

### INT 61h Function 0003h
*OPTIMA 1024 VGA-Sync,ET-3000 chipset*

### *REPORT ZOOM FACTOR*

**Purpose:** Returns the current zoom factor.

**Registers at call:**
AX = 0003h

**Return Registers:**
AX = zoom factor

**Conflicts:** see Chapter 2

**See also:** Functions 0000h and 0005h, INT 16h Function FFh

---

### INT 61h Function 0004h
*OPTIMA 1024 VGA-Sync,ET-3000 chipset*

### *ENTER SPECIFY MODE*

**Registers at call:**
AX = 0004h

**Return Registers:**
none

**Details:** Specify Mode is enabled by hot key (see also Function 0007h), and allows panning and zooming via the numeric keypad.
**Conflicts:** see Chapter 2
**See also:** Functions 0000h and 0007h, INT 16h Function FFh

---

## INT 61h Function 0005h

*OPTIMA 1024 VGA-Sync*

### *QUERY ZOOM WINDOW*

**Registers at call:**
AX = 0005h
BX:CX -> buffer for window parameters (see Table 9-22)

**Return Registers:**
none

**Conflicts:** see Chapter 2
**See also:** Functions 0000h, 0003h, and 0006h; INT 16h Function FFh

**9-22 Format of window parameters:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | X start of zoom window |
| 02h | WORD | Y start of zoom window |
| 04h | WORD | X end of zoom window |
| 06h | WORD | Y end of zoom window |
| 08h | WORD | current zoom factor |
| 0Ah | WORD | zoom offset start X |
| 0Ch | WORD | zoom offset start Y |

---

## INT 61h Function 0006h

*OPTIMA 1024 VGA-Sync,ET-3000 chipset*

### *SET ZOOM WINDOW*

**Registers at call:**
AX = 0006h
BX:CX -> zoom window description (see Table 9-23)

**Return Registers:**
none

**Details:** The width of the zoom window must be a multiple of the pixel replication (zoom) factor.
**Conflicts:** see Chapter 2
**See also:** Functions 0000h, 0001h, 0005h, and 0008h; INT 16h Function FFh

**9-23 Format of zoom window description:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | X start of zoom window |
| 02h | WORD | Y start of zoom window |
| 04h | WORD | X end of zoom window |
| 06h | WORD | Y end of zoom window |

---

## INT 61h Function 0007h

*OPTIMA 1024 VGA-Sync,ET-3000 chipset*

### *QUERY APPLICATION KEY*

**Purpose:** Returns the current toggle state of the application (END) key in specify mode. E.g. in the OPTIMA AutoCAD driver, 0 means AutoCAD calls INT 61h Function 0001h every time the crosshair cursor moves.

**Registers at call:**
AX = 0007h

**Return Registers:**
AX = current state (0000h/0001h)

**Details:** In specify mode, the END key has been reserved for applications.
**Conflicts:** see Chapter 2
**See also:** Functions 0000h, 0001h, and 0004h; INT 16h Function FFh; AutoCAD INT 7Ah (Chapter 66)

---

## INT 61h Function 0008h

*OPTIMA 1024 VGA-Sync,ET-3000 chipset*

### *SET ZOOM OFFSET*

**Purpose:** Specify the first byte of video memory to appear in the zoom window.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 0008h | none |
| BX = X start of zoom offset | |
| CX = Y start of zoom offset | |

**Conflicts:** see Chapter 2
**See also:** Functions 0000h and 0006h, INT 16h Function FFh

## Miscellaneous

---

### INT 10h Function 10E0h

*Diamond Speedstar 24*

### *SET 24-BIT GRAPHICS MODE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 10E0h | *unknown* |
| BL = video mode (see also Tables 8-2, 8-27, and 9-4)<br>2Eh = 640x480 | |

**See also:** Function 10F0h, standard Function 00h (Chapter 8)

---

### INT 10h Function 1Dh

*SpeedSTAR Plus BIOS v4.23+*

### *SET SYNC PARAMETERS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 1Dh | none |
| AL = video mode | |
| ES = caller's segment | |

**Details:** The caller's segment contains at offset 5Ch (FCB field in PSP) or 100h a table with sync parameters (BIOS looks at both offsets) (see Table 9-24).

**9-24 Format of sync table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 9 BYTEs | ID contains string 'ey5CENTER' |
| 09h | 5 BYTEs | sync parameters for 640x480 modes 11h,12h,25h,26h,2Eh |
| 0Bh | 5 BYTEs | sync parameters for 800x600 modes 29h,30h,2Ah |
| 13h | 5 BYTEs | sync parameters for 1024x768 modes 37h,38h |

---

### INT 6Dh

*ATI VGA Wonder*

### *VIDEO BIOS ENTRY POINT*

**Purpose:** This vector points at the original INT 10h entry point set up by the ATI BIOS.
**Conflicts:** IBM VGA (Chapter 8), Trident VGA (above)
**See also:** INT 10h (Chapter 8)

CHAPTER 10

# *Low-Level Disk I/O*

## Standard BIOS

### INT 13h Function 00h
### *RESET DISK SYSTEM*

**Purpose:** Force the controller to recalibrate the drive heads (i.e. seek to track 0).

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 00h | AH = status (see Table 10-1) |
| DL = drive (if bit 7 is set both hard disks and floppy disks reset) | CF clear if successful (returned status = 00h) |
| | CF set on error |

**See also:** Functions 0Dh and 11h, TI Professional INT 4Eh (below), MS-DOS INT 21h Function 0Dh (Chapter 24)

### INT 13h Function 01h
### *GET STATUS OF LAST OPERATION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 01h | CF clear if successful (returned status = 00h) |
| DL = drive (bit 7 set for hard disk) | CF set on error |
| | AH = status of previous operation (see Table 10-1) |

**Details:** Some BIOSes return the status in AL; the PS/2 Model 30/286 returns the status in both AH and AL.

**10-1 Disk operation status:**

| | |
|---|---|
| 00h | successful completion |
| 01h | invalid function in AH or invalid parameter |
| 02h | address mark not found |
| 03h | disk write-protected |
| 04h | sector not found/read error |
| 05h | reset failed (hard disk) |
| 06h | disk changed (floppy) |
| 07h | drive parameter activity failed (hard disk) |
| 08h | DMA overrun |
| 09h | attempted DMA across 64K boundary |
| 0Ah | bad sector detected (hard disk) |
| 0Bh | bad track detected (hard disk) |
| 0Ch | unsupported track or invalid media |
| 0Dh | invalid number of sectors on format (hard disk) |
| 0Eh | control data address mark detected (hard disk) |

| | |
|---|---|
| 0Fh | DMA arbitration level out of range (hard disk) |
| 10h | uncorrectable CRC or ECC error on read |
| 11h | data ECC corrected (hard disk) |
| 20h | controller failure |
| 40h | seek failed |
| 80h | timeout (not ready) |
| AAh | drive not ready (hard disk) |
| BBh | undefined error (hard disk) |
| CCh | write fault (hard disk) |
| E0h | status register error (hard disk) |
| FFh | sense operation failed (hard disk) |

---

## INT 13h Function 02h
### *READ SECTOR(S) INTO MEMORY*

**Registers at call:**
AH = 02h
AL = number of sectors to read (must be nonzero)
CH = low eight bits of cylinder number
CL = sector number 1-63 (bits 0-5)
 high two bits of cylinder (bits 6-7, hard disk only)
DH = head number
DL = drive number (bit 7 set for hard disk)
ES:BX -> data buffer

**Return Registers:**
CF set on error
 if AH = 11h (corrected ECC error), AL = burst length
CF clear if successful
AH = status (see Table 10-1)
AL = number of sectors transferred

**Details:** Errors on a floppy may be due to the motor failing to spin up quickly enough; the read should be retried at least three times, resetting the disk with Function 00h between attempts.

The IBM AT BIOS and many other BIOSes use only the low four bits of DH (head number) since the WD-1003 controller which is the standard AT controller (and the controller that IDE emulates) only supports 16 heads.

AWARD AT BIOS and AMI 386sx BIOS have been extended to handle more than 1024 cylinders by placing bits 10 and 11 of the cylinder number into bits 6 and 7 of DH.

**See also:** Functions 03h and 0Ah

---

## INT 13h Function 03h
### *WRITE DISK SECTOR(S)*

**Registers at call:**
AH = 03h
AL = number of sectors to write (must be nonzero)
CH = low eight bits of cylinder number
CL = sector number 1-63 (bits 0-5)
 high two bits of cylinder (bits 6-7, hard disk only)
DH = head number
DL = drive number (bit 7 set for hard disk)
ES:BX -> data buffer

**Return Registers:**
CF set on error
CF clear if successful
AH = status (see Table 10-1)
AL = number of sectors transferred

**Details:** Errors on a floppy may be due to the motor failing to spin up quickly enough; the write should be retried at least three times, resetting the disk with Function 00h between attempts.

The IBM AT BIOS and many other BIOSes use only the low four bits of DH (head number) since the WD-1003 controller which is the standard AT controller (and the controller that IDE emulates) only supports 16 heads.

AWARD AT BIOS and AMI 386sx BIOS have been extended to handle more than 1024 cylinders by placing bits 10 and 11 of the cylinder number into bits 6 and 7 of DH.
**See also:** Functions 02h and 0Bh

---

### INT 13h Function 04h
### *VERIFY DISK SECTOR(S)*

**Registers at call:**
AH = 04h
AL = number of sectors to verify (must be nonzero)
CH = low eight bits of cylinder number
CL = sector number 1-63 (bits 0-5)
high two bits of cylinder (bits 6-7, hard disk only)
DH = head number
DL = drive number (bit 7 set for hard disk)
ES:BX -> data buffer (PC,XT,AT with BIOS prior to 11/15/85)

**Return Registers:**
CF set on error
CF clear if successful
AH = status (see Table 10-1)
AL = number of sectors verified

**Details:** Errors on a floppy may be due to the motor failing to spin up quickly enough; the write should be retried at least three times, resetting the disk with Function 00h between attempts.

This function does not compare the disk with memory, it merely checks whether the sector's stored CRC matches the data's actual CRC.

The IBM AT BIOS and many other BIOSes use only the low four bits of DH (head number) since the WD-1003 controller which is the standard AT controller (and the controller that IDE emulates) only supports 16 heads.

AWARD AT BIOS and AMI 386sx BIOS have been extended to handle more than 1024 cylinders by placing bits 10 and 11 of the cylinder number into bits 6 and 7 of DH.
**See also:** Function 02h

---

### INT 13h Function 05h
*FLOPPY DISK*
### *FORMAT TRACK*

**Registers at call:**
AH = 05h
AL = number of sectors to format
CH = track number
DH = head number
DL = drive number
ES:BX -> address field buffer (see Table 10-2)

**Return Registers:**
CF set on error
CF clear if successful
AH = status (see Table 10-1)

**Details:** On AT or higher, call Function 17h first. The number of sectors per track is read from the diskette parameter table pointed at by INT 1Eh.
**Conflicts:** Fixed Disk below, Future Domain SCSI (Chapter 11)
**See also:** Fixed Disk Function 05h, Functions 17h and 18h, INT 1Eh

**10-2 Format of address field buffer entry (one per sector in track):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | track number |
| 01h | BYTE | head number (0-based) |
| 02h | BYTE | sector number |
| 03h | BYTE | sector size (00h=128 bytes, 01h=256 bytes, 02h=512, 03h=1024) |

---

### INT 13h Function 05h
*FIXED DISK*
### *FORMAT TRACK*

**Registers at call:**
AH = 05h
AL = interleave value (XT-type controllers only)
ES:BX -> 512-byte format buffer

the first 2*(sectors/track) bytes contain F,N for each sector
F = 00h for good sector, 80h for bad sector
N = sector number
CH = cylinder number (bits 8,9 in high bits of CL)
CL = high bits of cylinder number (bits 7,6)
DH = head
DL = drive

**Return Registers:**
AH = status code (see Table 10-1)

**Details:** AWARD AT BIOS and AMI 386sx BIOS have been extended to handle more than 1024 cylinders by placing bits 10 and 11 of the cylinder number into bits 6 and 7 of DH.

For XT-type controllers on an AT or higher, Function 0Fh should be called first.

The IBM AT BIOS and many other BIOSes use only the low four bits of DH (head number) since the WD-1003 controller which is the standard AT controller (and the controller that IDE emulates) only supports 16 heads.

**Conflicts:** Floppy Disk above, Future Domain SCSI (Chapter 11)
**See also:** Floppy Disk Function 05h, Fixed Disk Functions 06h and 07h, Function 0Fh and 18h, ESDI Function 1Ah (below)

---

## INT 13h Function 06h
## *FORMAT TRACK AND SET BAD SECTOR FLAGS*

*FIXED DISK*
*IBM XT, IBM Portable*

**Registers at call:**
AH = 06h
AL = interleave value
CH = cylinder number (bits 8,9 in high bits of CL)
CL = sector number
DH = head
DL = drive

**Return Registers:**
AH = status code (see Table 10-1)

**Details:** AWARD AT BIOS and AMI 386sx BIOS have been extended to handle more than 1024 cylinders by placing bits 10 and 11 of the cylinder number into bits 6 and 7 of DH.
**Conflicts:** Future Domain SCSI (Chapter 11), Adaptec AHA-154xA (Chapter 11)
**See also:** Fixed Disk Functions 05h and 07h

---

## INT 13h Function 07h
## *FORMAT DRIVE STARTING AT GIVEN TRACK*

*FIXED DISK*
*IBM XT, IBM Portable*

**Registers at call:**
AH = 07h
AL = interleave value (XT only)
ES:BX = 512-byte format buffer (see Table 10-2)
CH = cylinder number (bits 8,9 in high bits of CL)
CL = sector number
DH = head
DL = drive

**Return Registers:**
AH = status code (see Table 10-1)

**Details:** AWARD AT BIOS and AMI 386sx BIOS have been extended to handle more than 1024 cylinders by placing bits 10 and 11 of the cylinder number into bits 6 and 7 of DH.
**Conflicts:** Future Domain SCSI (Chapter 11)
**See also:** Fixed Disk Functions 05h and 06h, ESDI Function 1Ah (below)

---

## INT 13h Function 08h
## *GET DRIVE PARAMETERS*

**Registers at call:**
AH = 08h
DL = drive (bit 7 set for hard disk)

**Return Registers:**
CF set on error
AH = status (07h) (see Table 10-1)

CF clear if successful
AH = 00h
BL = drive type (AT/PS2 floppies only) (see Table 10-3)
CH = low eight bits of maximum cylinder number
CL = maximum sector number (bits 5-0)
high two bits of maximum cylinder number (bits 7-6)
DH = maximum head number
DL = number of drives
ES:DI -> drive parameter table (floppies only)

**Details:** May return successful even though specified drive is greater than the number of attached drives of that type (floppy/hard); check DL to ensure validity.

For systems predating the IBM AT, this call is only valid for hard disks, as it is implemented by the hard disk BIOS rather than the ROM BIOS.

Toshiba laptops with HardRAM return DL=02h when called with DL=80h, but fail on DL=81h. The BIOS data at 40h:75h correctly reports 01h.

**See also:** Function 15h, INT 1Eh, INT 41h, Adaptec Function 06h (Chapter 11)

**10-3 Drive types:**

| | |
|---|---|
| 01h | 360K |
| 02h | 1.2M |
| 03h | 720K |
| 04h | 1.44M |
| 05h | (*reportedly an obscure drive type shipped on some IBM machines*)<br>2.88M on some newer BIOSes such as the AMI 486 BIOS |
| 06h | 2.88M |

---

## INT 13h Function 09h

*HARD DISK*

### *INITIALIZE CONTROLLER WITH DRIVE PARAMETERS*

*PC/AT, PS/2*

**Registers at call:**
AH = 09h
DL = drive (80h for first, 81h for second)

**Return Registers:**
CF clear if successful
CF set on error
AH = status (see Table 10-1)

**Details:** On the PC and XT, this function uses the parameter table pointed at by INT 41h.

On the AT and later, this function uses the parameter table pointed at by INT 41h if DL=80h, and the parameter table pointed at by INT 46h if DL=81h.

**See also:** INT 41h, INT 46h

---

## INT 13h Function 0Ah

*HARD DISK*

### *READ LONG SECTOR(S)*

*PC/AT and later models*

**Registers at call:**
AH = 0Ah
AL = number of sectors
CH = low eight bits of cylinder number
CL = sector number (bits 5-0)
high two bits of cylinder number (bits 7-6)
DH = head number
DL = drive number (80h = first, 81h = second)
ES:BX -> data buffer

**Return Registers:**
CF clear if successful
CF set on error
AH = status (see Table 10-1)
AL = number of sectors transferred

**Details:** This function reads in four to seven bytes of error-correcting code along with each sector's worth of information.

Data errors are not automatically corrected, and the read is aborted after the first sector with an ECC error. Used for diagnostics only on PS/2 systems.
**See also:** Functions 02h and 0Bh

---

### INT 13h Function 0Bh
### *WRITE LONG SECTOR(S)*

*HARD DISK*
*PC/AT and later models*

**Registers at call:**
AH = 0Bh
AL = number of sectors
CH = low eight bits of cylinder number
CL = sector number (bits 5-0)
high two bits of cylinder number (bits 7-6)
DH = head number
DL = drive number (80h = first, 81h = second)
ES:BX -> data buffer

**Return Registers:**
CF clear if successful
CF set on error
AH = status (see Table 10-1)
AL = number of sectors transferred

**Details:** Each sector's worth of data must be followed by four to seven bytes of error-correction information.

Used for diagnostics only on PS/2 systems.
**See also:** Functions 03h and 0Ah

---

### INT 13h Function 0Ch
### *SEEK TO CYLINDER*

*HARD DISK*

**Registers at call:**
AH = 0Ch
CH = low eight bits of cylinder number
CL = sector number (bits 5-0)
high two bits of cylinder number (bits 7-6)
DH = head number
DL = drive number (80h = first, 81h = second hard disk)

**Return Registers:**
CF set on error
CF clear if successful
AH = status (see Table 10-1)

**See also:** Functions 00h, 02h, and 0Ah

---

### INT 13h Function 0Dh
### *RESET HARD DISKS*

*HARD DISK*

**Registers at call:**
AH = 0Dh
DL = drive number (80h = first, 81h = second hard disk)

**Return Registers:**
CF set on error
CF clear if successful
AH = status (see Table 10-1)

**Details:** Reinitializes the hard disk controller, resets the specified drive's parameters, and recalibrates the drive's heads (seek to track 0).

Not for PS/2 ESDI drives.
**See also:** Function 00h, MS-DOS INT 21h Function 0Dh (Chapter 24)

---

### INT 13h Function 0Eh
### *READ SECTOR BUFFER*

*HARD DISK*
*PC/XT only*

**Registers at call:**
AH = 0Eh
DL = drive number (80h = first, 81h = second hard disk)
ES:BX -> buffer

**Return Registers:**
CF set on error
CF clear if successful
AH = status code (see Table 10-1)

**Details:** Transfers controller's sector buffer. No data is read from the drive.

Used for diagnostics only on PS/2 systems.
**See also:** Function 0Ah

---

### INT 13h Function 0Fh
### *WRITE SECTOR BUFFER*

*HARD DISK*
*PC/XT only*

**Registers at call:**
AH = 0Fh
DL = drive number (80h = first, 81h = second hard disk)
ES:BX -> buffer

**Return Registers:**
CF set on error
CF clear if successful
AH = status code (see Table 10-1)

**Details:** Does not write data to the drive.
Should be called before formatting to initialize an XT-type controller's sector buffer.
Used for diagnostics only on PS/2 systems.

**See also:** Function 0Bh

---

### INT 13h Function 10h
### *CHECK IF DRIVE READY*

*HARD DISK*

**Registers at call:**
AH = 10h
DL = drive number (80h = first, 81h = second hard disk)

**Return Registers:**
CF set on error
CF clear if successful
AH = status (see Table 10-1)

---

### INT 13h Function 11h
### *RECALIBRATE DRIVE*

*HARD DISK*

**Registers at call:**
AH = 11h
DL = drive number (80h = first, 81h = second hard disk)

**Return Registers:**
CF set on error
CF clear if successful
AH = status (see Table 10-1)

**Details:** Causes hard disk controller to seek the specified drive to cylinder 0.

**See also:** Functions 00h and 0Ch, Fixed Disk Function 19h

---

### INT 13h Function 12h
### *CONTROLLER RAM DIAGNOSTIC*

*HARD DISK*
*PC/XT, PS/2*

**Registers at call:**
AH = 12h
DL = drive number (80h = first, 81h = second hard disk)

**Return Registers:**
CF set on error
CF clear if successful
AH = status code (see Table 10-1)

**Conflicts:** Future Domain SCSI (Chapter 11)

**See also:** Functions 13h and 14h

---

### INT 13h Function 13h
### *DRIVE DIAGNOSTIC*

*HARD DISK*
*PC/XT, PS/2*

**Registers at call:**
AH = 13h
DL = drive number (80h = first, 81h = second hard disk)

**Return Registers:**
CF set on error
CF clear if successful
AH = status code (see Table 10-1)

**See also:** Functions 12h and 14h

---

### INT 13h Function 14h
### *CONTROLLER INTERNAL DIAGNOSTIC*

*HARD DISK*

**Registers at call:**
AH = 14h

**Return Registers:**
CF set on error
CF clear if successful
AH = status code (see Table 10-1)

**See also:** Functions 12h and 13h

### INT 13h Function 15h
### *GET DISK TYPE*

*IBM XT 1/10/86 or later, all newer models*

**Registers at call:**
AH = 15h
DL = drive number (bit 7 set for hard disk)

**Return Registers:**
CF clear if successful
  AH = type code
    00h no such drive
    01h floppy without change-line support
    02h floppy with change-line support
    03h hard disk
      CX:DX = number of 512-byte sectors
CF set on error
  AH = status (see Table 10-1)

**See also:** Functions 08h, 16h, and 17h, SCSI Function 19h (Chapter 11)

---

### INT 13h Function 16h
### *DETECT DISK CHANGE (XT 1/10/86 or later, XT286, AT, PS)*

*FLOPPY DISK*

**Registers at call:**
AH = 16h
DL = drive number

**Return Registers:**
CF clear if change line inactive
  AH = 00h (disk not changed)
CF set if change line active
  AH = status
    06h change line active or not supported
    80h drive not ready or not present

**Details:** Call Function 15h first to determine whether the drive supports a change line.

**See also:** Function 15h

---

### INT 13h Function 17h
### *SET DISK TYPE FOR FORMAT (AT,PS)*

*FLOPPY DISK*

**Registers at call:**
AH = 17h
AL = format type
  01h = 320/360K disk in 360K drive
  02h = 320/360K disk in 1.2M drive
  03h = 1.2M disk in 1.2M drive
  04h = 720K disk in 720K drive
DL = drive number

**Return Registers:**
CF set on error
CF clear if successful
AH = status (see Table 10-1)

**Details:** This function does not handle 1.44M drives; use Function 18h instead.

**Conflicts:** Future Domain SCSI (Chapter 11)

**See also:** Functions 15h and 18h

---

### INT 13h Function 18h
### *SET MEDIA TYPE FOR FORMAT*

*IBM AT model 3x9, all newer machines*

**Registers at call:**
AH = 18h
DL = drive number
CH = lower 8 bits of highest cylinder number (number of cylinders - 1)
CL = sectors per track (bits 0-5)
  top 2 bits of highest cylinder number (bits 6,7)

**Return Registers:**
AH = status
  00h requested combination supported
  01h function not available
  0Ch not supported or drive type unknown
  80h there is no disk in the drive
ES:DI -> 11-byte parameter table

**Conflicts:** Future Domain SCSI (Chapter 11)

**See also:** Functions 05h, 07h, and 17h

---

## INT 13h Function 19h

FIXED DISK

### *PARK HEADS (XT286,PS)*

**Registers at call:**
AH = 19h
DL = drive

**Return Registers:**
CF set on error
AH = status (see Table 10-1)

**See also:** Function 11h

---

## INT 1Eh

SYSTEM DATA

### *DISKETTE PARAMETERS*

**Details:** Default parameter table at F000h:EFC7h for 100% compatible BIOSes.
**See also:** INT 13h Function 0Fh, INT 41h

**10-4 Format of diskette parameter table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | first specify byte |
| | | bits 7-4: step rate |
| | | bits 3-0: head unload time (0Fh = 240 ms) |
| 01h | BYTE | second specify byte |
| | | bits 7-1: head load time (01h = 4 ms) |
| | | bit0: non-DMA mode (always 0) |
| 02h | BYTE | delay until motor turned off (in clock ticks) |
| 03h | BYTE | bytes per sector (00h = 128, 01h = 256, 02h = 512, 03h = 1024) |
| 04h | BYTE | sectors per track |
| 05h | BYTE | length of gap between sectors (2Ah for 5.25” , 1Bh for 3.5” ) |
| 06h | BYTE | data length (ignored if bytes-per-sector field nonzero) |
| 07h | BYTE | gap length when formatting (50h for 5.25” , 6Ch for 3.5” ) |
| 08h | BYTE | format filler byte (default F6h) |
| 09h | BYTE | head settle time in milliseconds |
| 0Ah | BYTE | motor start time in 1/8 seconds |

---

## INT 40h

DISKETTE

### *ROM BIOS DISKETTE HANDLER RELOCATED BY HARD DISK BIOS*

**See also:** INT 13h, SuperBIOS INT 47h (below), 4+Power INT 63h (below)

---

## INT 41h

SYSTEM DATA

### *HARD DISK 0 PARAMETER TABLE*

**Details:** The default parameter table array is located at F000h:E401h in 100% compatible BIOSes; the pointer may be overridden by the hard disk controller’s BIOS to support drive formats unknown to the ROM BIOS.
**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), MS Windows debugging kernel (Chapter 57)
**See also:** INT 13h Function 09h, INT 1Eh, INT 46h

**10-5 Format of fixed disk parameters:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of cylinders |
| 02h | BYTE | number of heads |
| 03h | WORD | starting reduced write current cylinder (XT only, 0 for others) |
| 05h | WORD | starting write precompensation cylinder number |
| 07h | BYTE | maximum ECC burst length (XT only) |
| 08h | BYTE | control byte |
| | | bits 0-2: drive option (XT only, 0 for others) |

| | | |
|---|---|---|
| | | bit 3: set if more than 8 heads (AT and later only) |
| | | bit 4: always 0 |
| | | bit 5: set if manufacturer's defect map on max cylinder+1 (AT and later only) |
| | | bit 6: disable ECC retries |
| | | bit 7: disable access retries |
| 09h | BYTE | standard timeout (XT only, 0 for others) |
| 0Ah | BYTE | formatting timeout (XT and WD1002 only, 0 for others) |
| 0Bh | BYTE | timeout for checking drive (XT and WD1002 only, 0 for others) |
| 0Ch | WORD | cylinder number of landing zone (AT and later only) |
| 0Eh | BYTE | number of sectors per track (AT and later only) |
| 0Fh | BYTE | reserved |

---

### INT 46h

*SYSTEM DATA*

#### *HARD DISK 1 DRIVE PARAMETER TABLE*

**Purpose:** This vector points at the parameter table (see Table 10-5) for the second hard disk, if present.

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7)

**See also:** INT 13h Function 09h, INT 41h

## ESDI Controllers

The Enhanced Small Device Interface is a higher-performance hard disk interface than the earlier ST-506 interface. Compared to the 5 megabit per second speed of the ST-506 interface, ESDI is up to four times faster, offering speeds of 10, 15, and 20 megabits per second. It has now effectively been replaced by IDE (Integrated Drive Electronics) drives which are cheaper and typically even faster than ESDI drives.

---

### INT 13h Function 1Ah

*ESDI FIXED DISK*

#### *FORMAT UNIT (PS)*

**Registers at call:**

AH = 1Ah
AL = defect table count
CL = format modifiers (see Table 10-6)
DL = drive
ES:BX -> defect table

**Return Registers:**

CF set on error
AH = status (see Table 10-1)

**Details:** If the periodic interrupt is selected, INT 15h Function 0Fh is called after each cylinder is formatted.

**Conflicts:** Future Domain SCSI (Chapter 11)

**See also:** INT 15h Function 0Fh, Standard Function 07h (above)

**10-6 Bitfields for format modifiers:**

| | |
|---|---|
| bit 4 | generate periodic interrupt |
| bit 3 | perform surface analysis |
| bit 2 | update secondary defect map |
| bit 1 | ignore secondary defect map |
| bit 0 | ignore primary defect map |

---

### INT 13h Function 1Bh

*ESDI FIXED DISK*

#### *GET MANUFACTURING HEADER*

**Registers at call:**

AH = 1Bh
AL = number of record
DL = drive

ES:BX -> buffer for manufacturing
  header (defect list)

**Return Registers:**
CF set on error
  AH = status

**Details:** The manufacturing header format (Defect Map Record format) can be found in the IBM 70MB, 115MB Fixed Disk Drives Technical Reference.
**Conflicts:** Future Domain SCSI (Chapter 11)

---

### INT 13h Function 1C0Ah

*ESDI FIXED DISK*

### *GET DEVICE CONFIGURATION*

**Registers at call:**
AX = 1C0Ah
DL = drive
ES:BX -> buffer for device configuration
  (drive physical parameter)

**Return Registers:**
CF set on error
  AH = status

**Details:** The device configuration format can be found in IBM ESDI Fixed Disk Drive Adapter/A Technical Reference.
**Conflicts:** Future Domain SCSI (Chapter 11)

### INT 13h Function 1C0Bh

*ESDI FIXED DISK*

### *GET ADAPTER CONFIGURATION*

**Registers at call:**
AX = 1C0Bh
ES:BX -> buffer for adapter configuration

**Return Registers:**
CF set on error
  AH = status

**Conflicts:** Future Domain SCSI (Chapter 11)
**See also:** Function 1C0Ch

### INT 13h Function 1C0Ch

*ESDI FIXED DISK*

### *GET POS INFORMATION*

**Registers at call:**
AX = 1C0Ch
ES:BX -> POS information

**Return Registers:**
CF set on error
  AH = status

**Conflicts:** Future Domain SCSI (Chapter 11)
**See also:** Function 1C0Bh

### INT 13h Function 1C0Eh

*ESDI FIXED DISK*

### *TRANSLATE RBA TO ABA*

**Registers at call:**
AX = 1C0Eh
CH = low 8 bits of cylinder number
CL = sector number, high two bits of
  cylinder number in bits 6 and 7
DH = head number
DL = drive number
ES:BX -> ABA number

**Return Registers:**
CF set on error
  AH = status

**Details:** The ABA (absolute block address) format can be found in the IBM ESDI Adapter Technical Reference by using its Device Configuration Status Block.
**Conflicts:** Future Domain SCSI (Chapter 11)

### INT 15h Function 0Fh

*SYSTEM*
*Callout*

### *FORMAT UNIT PERIODIC INTERRUPT (PS ESDI drives only)*

**Registers at call:**
AH = 0Fh
AL = phase code
    00h reserved

01h surface analysis
02h formatting

**Return Registers:**
CF clear if formatting should continue, set if it should terminate

**Details:** Called during ESDI drive formatting after each cylinder is completed.
**Conflicts:** MultiDOS Plus (Chapter 45), VMiX (Chapter 46)
**See also:** INT 13h Function 1Ah

## Vendor-Specific Extensions

---

### INT 13h Function 20h
*Western Digital " Super BIOS"*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AH = 20h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Returns some kind of status related to whether the drive contains its default media type.
**Conflicts:** QuickCache II (Chapter 17)

---

### INT 13h Function 5504h
*Seagate*

### *GET CONTROLLER TYPE*

**Registers at call:**
AX = 5504h
DX = drive (bit 7 set for hard disk)

**Return Registers:**
CF clear if successful
    AX = 4321h,4322h if Seagate controller
CF set on error

**See also:** Functions 5505h and 5514h

---

### INT 13h Function 5505h
*Seagate*

### *PARK HEADS*

**Registers at call:**
AX = 5505h
DX = drive (bit 7 set for hard disk)

**Return Registers:**
CF clear if successful
CF set on error

**See also:** Functions 5504h and 5515h

---

### INT 13h Function 5514h
*Seagate*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 5514h
DX = drive (bit 7 set for hard disk)

**Return Registers:**
CF clear if successful
CF set on error
AX = return value (FEBEh,FEBFh,FEDAh,FEDBh)

**See also:** Functions 5504h and 5515h

---

### INT 13h Function 5515h
*Seagate*

### *PARK HEADS*

**Registers at call:**
AX = 5515h
DX = drive (bit 7 set for hard disk)

**Return Registers:**
CF clear if successful
CF set on error

**Details:** This function appears to be identical to Function 5505h.
**See also:** Functions 5504h and 5505h

---

### INT 13h Function 70h
*Priam EDVR.SYS*

### *DISK PARTITIONING SOFTWARE*

**Registers at call:**
AH = 70h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Priam's EDISK.EXE (FDISK replacement) and EFMT.EXE (low-level formatting program) make this call, presumably to EDVR.SYS (the partitioning driver).
**See also:** Function ADh

---

### INT 13h Function ADh *Priam HARD DISK CONTROLLER*
### *UNDETERMINED FUNCTION*

| Registers at call: | Return Registers: |
|---|---|
| AH = ADh | *unknown* |
| *others, if any, unknown* | |

**Details:** This call is made from Priam's EFMT.EXE (low-level formatter), probably to check the ROM type on the controller for their hard disk kits.
**Conflicts:** Super PC-Kwik (Chapter 17)
**See also:** Function 70h

### INT 13h Function EEh *SWBIOS*
### *SET 1024 CYLINDER FLAG*

| Registers at call: | Return Registers: |
|---|---|
| AH = EEh | CF clear AH = 00h |
| DL = drive number (80h, 81h) | |

**Program:** SWBIOS is a TSR by Ontrack Computer Systems
**Details:** The following INT 13h call will interpret the cylinder number as 1024 less than the desired cylinder.

Flag cleared by all INT 13h calls except Function EEh.

Disk Manager also supports these calls.

Also supported by HyperDisk v4.01+ and PC-Cache v5.5+, in order to allow caching of drives using SWBIOS to access more than 1024 cylinders.

**See also:** Functions F9h and FEh, INT 16h Function FFA5h Subfunction 1111h (Chapter 17), INT 2Fh Function DFh (Chapter 17)

### INT 13h Function F9h *SWBIOS*
### *INSTALLATION CHECK*

| Registers at call: | Return Registers: |
|---|---|
| AH = F9h | CF clear |
| DL = drive number (80h,81h) | DX = configuration word<br>bit 15 set if other SWBIOS extensions available |
| | CF set on error |

**Program:** SWBIOS is a TSR by Ontrack Computer Systems.
**Details:** Disk Manager also supports these calls.
**See also:** Function EEh

### INT 13h Function FEh *SWBIOS*
### *GET EXTENDED CYLINDER COUNT*

| Registers at call: | Return Registers: |
|---|---|
| AH = FEh | CF clear |
| DL = drive number (80h, 81h) | DX = number of cylinders beyond 1024 on drive |

**Program:** SWBIOS is a TSR by Ontrack Computer Systems
**Details:** Standard INT 13h Function 08h will return a cylinder count truncated to 1024.

A BIOS without this extension would return the count modulo 1024.

Disk Manager also supports these calls.

**See also:** Function EEh

---

### INT 2Fh Function 57h

*Iomega (undocumented)*

**Registers at call:**
AH = 57h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Reportedly used by Iomega Corp.
**Conflicts:** none, but see Table 2-2

---

### INT 2Fh Function C000h
### *INSTALLATION CHECK*

*WANG_ER.COM (undocumented)*

**Registers at call:**
AX = C000h

**Return Registers:**
AL = FFh if installed
ES = segment of resident code

**Program:** WANG_ER is a TSR from Compaq which permits Compaq systems equipped with 3-mode floppy drives to read Wang document diskettes
**Details:** AH=C0h is the default; WANG_ER scans all multiplex numbers from C0h to FFh for a free spot on installation. The installation check is completed by comparing the resident code with the copy in the transient program.
**Conflicts:** see Table 2-2

---

### INT 42h
### *INT 40h CASCADE*

*Western Digital WD1002 SuperBIOS*

**Details:** If the second WD1002 controller in the system finds INT 40h already in use, it uses this vector to cascade to the first controller's BIOS.
**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), EGA/VGA Video (Chapter 8)
**See also:** INT 47h, Standard INT 40h (above)

---

### INT 47h
### *INT 40h CASCADE*

*Western Digital WD1002-27X SuperBIOS*

**Purpose:** Used by the second WD1002-27X controller to cascade to the first controller's INT 40h.
**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), SQL Base (Chapter 54)
**See also:** INT 48h, Standard INT 40h and INT 42h (above)

---

### INT 48h
### *DRIVE DATA (NOT A VECTOR!)*

*Western Digital WD1002-27X SuperBIOS*

**Details:** The second WD1002-27X controller in a system uses the low byte to store the number of drives controlled by the second controller, and the high word for temporary storage during track recalculation; the first controller uses offsets 74h-77h in the BIOS data area (see MEMORY.LST) to store data.
**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), PCjr (Chapter 13), Compaq UILIB (Chapter 54), Watstar PC Network (NI)
**See also:** INT 47h

---

### INT 4Eh
### *DISK I/O*

*TI Professional PC*

**Purpose:** This interrupt is used instead of INT 13h on the TI Professional PC.
**Conflicts:** Z100 (Chapter 7)
**See also:** Standard INT 13h (above)

**INT 60h** *Adaptec and OMTI controllers*

***DRIVE 0 DATA***

**Purpose:** This vector stores the first four bytes of the parameter table for hard disk 0.
**Conflicts:** see Table 2-3
**See also:** Adaptec INT 61h through INT 64h
**Details:** These vectors are used by the following Adaptec controllers:

ACB 2370 A/B/C, ACB 2372 A/B/C, ACB 2333 A/B, 2322B-8, 2322B-16.

These vectors are NOT used by the following Adaptec controllers:

ACB 2310, ACB 2312, ACB 2320D, ACB 2322D.

**INT 61h** *Adaptec and OMTI controllers*

***DRIVE 0 DATA***

**Purpose:** This vector stores the second four bytes of the parameter table for hard disk 0.
**Conflicts:** see Table 2-3
**See also:** Adaptec INT 60h, INT 62h, and INT 63h

**INT 62h** *Adaptec and OMTI controllers*

***DRIVE 0 DATA***

**Purpose:** This vector stores the third four bytes of the parameter table for hard disk 0.
**Conflicts:** see Table 2-3
**See also:** Adaptec INT 60h, INT 61h, and INT 63h

**INT 63h** *Adaptec and OMTI controllers*

***DRIVE 0 DATA***

**Purpose:** This vector stores the last four bytes of the parameter table for hard disk 0.
**Conflicts:** see Table 2-3
**See also:** Adaptec INT 60h, INT 61h, INT 62h, and INT 64h

**INT 63h** *4+Power FLOPPY CONTROLLER*

***ORIGINAL INT 13h/40***

**Purpose:** The " 4+Power" quad floppy controller BIOS hooks INT 13h (or INT 40h if INT 13h has been moved there) and places the old value here.
**Conflicts:** see Table 2-3

**INT 64h** *Adaptec controllers*

***DRIVE 1 DATA***

**Purpose:** This vector stores the first four bytes of the parameter table for hard disk 1.
**Details:** These vectors are used by the following Adaptec controllers:

ACB 2370 A/B/C, ACB 2372 A/B/C, ACB 2333 A/B, 2322B-8, 2322B-16.

These vectors are NOT used by the following Adaptec controllers:

ACB 2310, ACB 2312, ACB 2320D, ACB 2322D.

**Conflicts:** see Table 2-3
**See also:** Adaptec INT 60h, INT 65h, INT 66h, and INT 67h

**INT 64h** *Pdisk by Scott Garfinkle*

***Overwritten for Hard Drive information***

**Purpose:** This vector is overwritten by Pdisk to install custom harddrive types. It can either destroy 4 vectors and take no memory or TSR and take up some memory.
**Conflicts:** see Table 2-3
**See also:** Pdisk INT 65h

---

**INT 65h** *Adaptec controllers*

***DRIVE 1 DATA***

**Purpose:** This vector stores the second four bytes of the parameter table for hard disk 1.
**Conflicts:** see Table 2-3
**See also:** Adaptec INT 64h, Adaptec INT 66h, Adaptec INT 67h

---

**INT 65h** *Pdisk by Scott Garfinkle*

***Overwritten for Hard Drive information***

**Conflicts:** see Table 2-3
**See also:** Pdisk INT 64h, Pdisk INT 66h

---

**INT 66h** *Adaptec controllers*

***DRIVE 1 DATA***

**Purpose:** This vector stores the third four bytes of the parameter table for hard disk 1.
**Conflicts:** see Table 2-3
**See also:** Adaptec INT 64h, Adaptec INT 65h, Adaptec INT 67h

---

**INT 66h** *Pdisk by Scott Garfinkle*

***Overwritten for Hard Drive information***

**Conflicts:** see Table 2-3
**See also:** Pdisk INT 64h, Pdisk INT 67h

---

**INT 67h** *Adaptec controllers*

***DRIVE 1 DATA***

**Purpose:** This vector stores the last four bytes of the parameter table for hard disk 1.
**Conflicts:** see Table 2-3
**See also:** Adaptec INT 64h, Adaptec INT 65h, Adaptec INT 66h

---

**INT 67h** *Pdisk by Scott Garfinkle*

***Overwritten for Hard Drive information***

**Conflicts:** see Table 2-3
**See also:** Pdisk INT 64h, Pdisk INT 66h

---

**INT 78h** *ADP-60 IDE controller*

***ORIGINAL INT 13h***

**Conflicts:** TARGA.DEV (Chapter 11), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)
**See also:** ADP-60 INT 79h

---

**INT 79h** *ADP-60 IDE adapter (undocumented)*

***UNDETERMINED FUNCTION***

**Conflicts:** AVATAR.SYS (Chapter 13), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD Device Interface (Chapter 66)
**See also:** ADP-60 INT 78h

## RAMdisks

The availability of a standard BIOS interface for disks which the operating system calls makes it fairly straight-forward to simulate a physical disk using RAM instead, and provide a BIOS-compatible interface. This is the approach used by the earliest RAMdisks, which were available even before MS-DOS version 2.0 introduced installable device drivers. Since the use of device drivers provides more flexibility, modern RAMdisks use device drivers instead of emulating INT 13h. xDISK is a shareware resizeable EMS RAMdisk by FM de Monasterio.

## INT 65h

### INT 21h Function 4404h
### *CONFIGURE*

*xDISK v3.31*
*(undocumented)*

**Registers at call:**
AX = 4404h
BL = drive number (00h = default, 01h = A:, etc)
CX = 0047h (length of version string)
DS:DX -> 79-byte buffer for version string and *unknown value*
DS:0081h = commandline containing new switches for driver

**Return Registers:**
CF clear if successful
AX = number of bytes actually transferred
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Conflicts:** Stacker (Chapter 16), DoubleDisk (Chapter 16), DoubleSpace (Chapter 16), COMBI-disk (Chapter 17)
**See also:** xDISK Function 4405h, INT 2Fh Function DDh

### INT 21h Function 4405h
### *UNDETERMINED FUNCTION*

*xDISK v3.31*
*(undocumented)*

**Registers at call:**
AX = 4405h
BL = drive number (00h = default, 01h = A:, etc)
CX = number of bytes to write
DS:DX -> buffer containing version string
*others, if any, unknown*

**Return Registers:**
CF clear if successful
AX = number of bytes actually transferred
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Conflicts:** ST-01 SCSI.SYS (Chapter 11), DoubleSpace (Chapter 16), COMBI-disk (Chapter 17)
**See also:** xDISK Function 4404h, INT 2Fh Function DDh

### INT 2Fh Function DDh
### *INSTALLATION CHECK*

*xDISK v3.32+*

**Registers at call:**
AH = DDh
BX = 7844h ('xD')
CX = 4953h ('IS')
DX = 4B3Fh ('K?')
AL = desired drive (01h-1Ah) or 00h to check for xDISK on any drive
ES:DI -> 25-byte data buffer (see Table 10-8)

**Return Registers:**
AX = DDFFh if installed (on specified drive if AL nonzero on entry)
BX = 87BBh
DX = B4C0h
ES:DI buffer filled
CX,CF destroyed

**Conflicts:** CappaCom programs (Chapter 51); see also Table 2-2
**See also:** xDISK INT 21h Functions 4404h and 4405h

**10-8 Format of data buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to ASCIZ driver signature " xDISK unit: X" |
| 04h | BYTE | flag: 01h if disk linked to DOS, 00h if unlinked |
| 05h | BYTE | flag: 01h if write protected, 00h if not |
| 06h | BYTE | flag: 01h if root directory full, 00h if not |
| 07h | BYTE | flag: 01h if free space uncompacted, 00h if compacted |
| 08h | BYTE | resizing state: 00h not resizable, 01h resized, 80h resizable |
| 09h | BYTE | flag: 01h inelastic resizable disk, 00h elastic |
| 0Ah | 2 BYTEs | reserved |
| 0Ch | BYTE | flag: 01h collapsed disk, 00h not collapsed |
| 0Dh | BYTE | flag: 01h using all EMS, 00h some EMS free |
| 0Eh | BYTE | flag: 01h password enabled, 00h disabled |

| | | |
|---|---|---|
| 0Fh | BYTE | flag: 01h password audio feedback, 00h no feedback |
| 10h | BYTE | flag: 01h password video feedback, 00h no feedback |
| 11h | BYTE | flag: 01h confirm changes, 00h no confirmation |
| 12h | BYTE | flag: 01h terse display, 00h verbose display |
| 13h | BYTE | flag: 01h click speaker on disk access, 00h no click |
| 14h | BYTE | flag: 01h flash icon on disk access, 00h no icon flash |
| 15h | BYTE | FAT entry size: 00h 12-bit, FFh 16-bit |
| 16h | WORD | count of open files in RAM disk |
| 18h | BYTE | unused |

## INT 2Fh Function DDh

# CHAPTER 11

# *SCSI Device I/O*

The Small Computer Systems Interface, better known as SCSI (pronounced "scuzzy"), has been on the scene since 1979 but is only now gaining widespread acceptance. Originally developed at Seagate to connect 8-inch Winchester drives to minicomputers, it became an ANSI standard (X3.131) in 1986. The original SCSI standard has been augmented by a newer version known as SCSI-2.

Both the original standard, and SCSI-2, specify the use of a SCSI Host Adapter to do the job done in older interfaces by a controller card. SCSI itself specifies only the interface between the host adapter and the peripheral device, making direct programming of SCSI devices more difficult than it had to be. To solve this problem, a number of programming interfaces have been developed. The most widely used are listed in this chapter.

## ASPI

ASPI is the Advanced SCSI Programming Interface by Adaptec, which has become a de facto standard for accessing SCSI devices.

---

### INT 21h Function 4402h *ASPI*

***INTERFACE***

**Registers at call:**
AX = 4402h
BX = file handle for device "SCSIMGR$"
CX = 0004h
DS:DX -> buffer for function address

**Return Registers:**
CF clear if successful
    AX = 0004h
CF set on error
    AX = error code (01h,05h,06h, 0Dh) (see Table 24-2)

**Details:** The function address is called with the address of a SCSI Request Block on the stack and the caller must clean up the stack.

**See also:** ASPITAPE Function 440Ch, WD7000 INT 11h Function FFh

**11-1 Format of SCSI Request Block (64 bytes):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | request number<br>00h "HA_INQ" host adapter inquiry<br>01h "GET_TYPE" get device type<br>02h "EXEC_SIO" execute SCSI I/O<br>03h "ABORT_SRB" abort SRB<br>04h "RESET_DEV" reset SCSI device<br>05h "SET_HAPRMS" set host adapter parameters |
| 01h | BYTE | request status |

| | | |
|---|---|---|
| | | 00h not done yet |
| | | 02h aborted |
| | | 04h SCSI I/O error |
| | | 80h invalid |
| | | 81h no adapter |
| | | 82h no device attached |
| | | else status |
| 02h | BYTE | host adapter ID |
| 03h | BYTE | request flags |
| | | bit 3: *unknown* |
| 04h | DWORD | reserved |
| ---**request 00h**--- | | |
| 08h | BYTE | (returned) number of host adapters |
| 09h | BYTE | (returned) target adapter ID |
| 0Ah | 16 BYTEs | (returned) manager ID |
| 1Ah | 16 BYTEs | (returned) adapter ID |
| 2Ah | 16 BYTEs | (returned) parameters |
| ---**request 01h**--- | | |
| 08h | BYTE | target ID |
| 09h | BYTE | logical unit number |
| 0Ah | BYTE | (returned) device type |
| | | 01h streamer |
| ---**request 02h**--- | | |
| 08h | BYTE | target ID |
| 09h | BYTE | logical unit number |
| 0Ah | DWORD | data allocation length |
| 0Eh | BYTE | sense allocation length |
| 0Fh | DWORD | data buffer pointer |
| 13h | DWORD | next request pointer (for linking) |
| 17h | BYTE | CDB length |
| 18h | BYTE | (returned) host adapter status |
| | | 11h select timeout |
| | | 12h data overrun |
| | | 13h bus error |
| | | 14h bus failure |
| 19h | BYTE | (returned) target status |
| | | 02h sense data stored in SRB |
| | | 08h target busy |
| | | 18h reservation error |
| 1Ah | DWORD | post routine address |
| 1Eh | WORD | real mode Post DS |
| 20h | DWORD | SRB pointer |
| 24h | WORD | reserved |
| 26h | DWORD | SRB physical address |
| 2Ah | 22 BYTEs | SCSIMGR$ workspace |
| 40h | N BYTEs | CCB (20-24 bytes) |
| ---**request 04h**--- | | |
| 08h | BYTE | target ID |
| 09h | BYTE | logical unit number |

## INT 21h Function 4402h

| | | |
|---|---|---|
| 0Ah | 14 BYTEs | reserved |
| 18h | BYTE | (returned) host adapter status (see above) |
| 19h | BYTE | (returned) target status (see above) |
| 1Ah | DWORD | post routine address |
| 1Eh | 34 BYTEs | workspace |

### 11-2 Format of CCB:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | command code<br>01h rewind<br>05h get block size limits<br>08h read<br>0Ah write<br>10h write file marks<br>11h SCSI Space (set position?)<br>12h SCSI Inquire<br>19h erase<br>1Bh load/unload media |
| 01h | BYTE | flags<br>bits 4-0: vary by function<br>bits 7-5: logical unit number |
| 02h | BYTE | "adr_1" |
| 03h | BYTE | "adr_0" |
| 04h | BYTE | length |
| 05h | BYTE | control |
| | ... | |
| 06h/0Ah | 14 BYTEs | buffer for sense data |

## INT 21h Function 440Ch

*Greg Shenaut ASPITAPE.SYS*

***INTERFACE***

**Registers at call:**

AX = 440Ch

BX = device handle (see below)

CH = category code

    07h tape (ASPITAPE.SYS)

CL = function

    01h "mtop" - perform tape operation

    02h "mtget" - get tape status

    03h ignore end-of-tape errors

    04h enable end-of-tape errors

DS:DX -> parameter block (see Table 11-3)

**Return Registers:**

CF set on error

    AX = error code (see Table 24-2)

CF clear if successful

    DS:DX -> data block

**Details:** This device driver is a simple DOS interface to the Adaptec Advanced SCSI Programming Interface (ASPI). It provides the following device names as access to the SCSI tape, 'RMTx' (rewind on close) and 'NRMTx' (NO rewind on close) where x can range from 0 to 3. There may also be devices named 'MTx' and 'NMTx' which default to 1024 byte blocks. The names may also have a '$' appended to try and make them unique from file names of 'RMT0' etc.

Once opened, these devices must be put into RAW mode.

**See also:** ASPI Function 4402h

**11-3 Format of mtop parameter block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | operation code<br>00h "MTWEOF" write an end-of-file record<br>01h "MTFSF" forward space file<br>02h "MTBSF" backward space file<br>03h "MTFSR" forward space record<br>04h "MTBSR" backward space record<br>05h "MTREW" rewind<br>06h "MTOFFL" rewind and unload<br>07h "MTNOP" perform TEST UNIT READY |
| 02h | DWORD | repetition count |

**11-4 Format of mtget parameter block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | ASPI host ID |
| 01h | BYTE | SCSI target ID |
| 02h | BYTE | SCSI logical unit number |
| 03h | BYTE | device parameters<br>bit 0: drive must use fixed-block read and write<br>bit 7: drive is an ASPI device |
| 04h | BYTE | current device state (see Table 11-5) |
| 05h | BYTE | unit number within driver |
| 06h | WORD | fixed block blocksize |
| 08h | BYTE | last SCSI status |
| 09h | BYTE | last SCSI sense key |
| 0Ah | WORD | last SCSI opcode (packed)<br>bits 0-7: SCSI operation (SCSI packet byte 0).<br>bits 8-10: SCSI flags (SCSI packet byte 1)<br>bits 11-12: ASPI "Direction Bits" (ASPI SRB byte 3) |
| 0Ch | WORD | residual bytes from SCSI opcode |

**11-5 Bitfields for current device state:**

| | |
|---|---|
| bit 0 | device currently opened in buffered mode |
| bit 1 | drive currently opened in nonbuffered mode |
| bit 2 | rewind drive on last close |
| bit 3 | drive has been written on |
| bit 4 | drive has been read from |
| bit 5 | next read will return 0 bytes |
| bit 6 | EOM will resemble EOF |
| bit 7 | drive may be busy rewinding |

## Common Access Method

The Common Access Method is a committee-defined SCSI interface which competes with ASPI.

**INT 4Bh**
***SCSI INTERFACE***

*CAM*
*draft revision 1.9*

**Registers at call:**
ES:DI -> CAM Control Block (see Table 11-7

**Return Registers:**
see INT 4Fh Function 8100h

**Details:** The CAM committee moved the interface to INT 4Fh after revision 1.9 to avoid conflicting with the IBM SCSI interface and the Virtual DMA specification.

The installation check for the driver is the string "SCSI_CAM" eight bytes past the INT 4Bh handler.

It is not known whether any drivers actually implemented this interface on INT 4Bh instead of INT 4Fh.

**Conflicts:** IBM SCSI interface (below), Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Virtual DMA Specification (Chapter 18)

**See also:** INT 4Fh Function 8100h

---

### INT 4Fh Function 8100h *CAM*
### ***SEND CCB TO XPT/SIM*** *revision 2.3*

**Registers at call:**
AX = 8100h
ES:BX -> CAM Control Block (CCB)
(see Table 11-7)

**Return Registers:**
AH = status
00h successful
01h invalid CCB address (0000h:0000h)

**Details:** The SCSI Interface Module (SIM) may complete the requested function and invoke the completion callback function before this call returns.

**Conflicts:** Z100 (Chapter 7)

**See also:** Function 8200h, INT 4Bh above, INT 2Fh Function 7F01h (below)

**11-6 CAM function codes:**

| | |
|---|---|
| 00h | NOP |
| 01h | execute SCSI I/O |
| 02h | get device type |
| 03h | path inquiry |
| 04h | release SIM queue |
| 05h | set async callback |
| 06h | set device type |
| 07h-0Fh | reserved |
| 10h | abort SCSI command |
| 11h | reset SCSI bus |
| 12h | reset SCSI device |
| 13h | terminate I/O process |
| 14h-1Fh | reserved |
| 20h | engine inquiry |
| 21h | execute engine request |
| 22h-2Fh | reserved |
| 30h | enable logical unit number |
| 31h | execute target I/O |
| 32h-7Fh | reserved |
| 80h-FFh | vendor-specific functions |

**11-7 Format of CAM Control Block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | physical address of this CCB |
| 04h | WORD | CAM control block length |
| 06h | BYTE | function code (see Table 11-6) |
| 07h | BYTE | CAM status (see Table 11-10) |
| 08h | BYTE | SCSI status |
| 09h | BYTE | path ID (FFh = XPT) |
| 0Ah | BYTE | target ID |

| | | |
|---|---|---|
| 0Bh | BYTE | logical unit number |
| 0Ch | WORD | CAM flags (see Table 11-8) |
| 0Eh | BYTE | CAM address flags (see Table 11-9) |
| 0Fh | BYTE | target-mode flags (see Table 11-11) |
| **---function 02h---** | | |
| 10h | DWORD | pointer to 36-byte buffer for inquiry data or 0000h:0000h |
| 14h | BYTE | peripheral device type of target logical unit number |
| **---function 03h---** | | |
| 10h | BYTE | version number (00h-07h prior to rev 1.7, 08h = rev 1.7, 09h-FFh = rev no, i.e. 23h = rev 2.3) |
| 11h | BYTE | SCSI capabilities (see Table 11-12) |
| 12h | BYTE | target mode support<br>bit 7: processor mode<br>bit 6: phase-cognizant mode<br>bit 5-0: reserved |
| 13h | BYTE | miscellaneous flags<br>bit 7: scanned high to low instead of low to high<br>bit 6: removables not included in scan<br>bit 5: inquiry data not kept by XPT<br>bits 4-0: reserved |
| 14h | WORD | engine count |
| 16h | 14 BYTEs | vendor-specific data |
| 24h | DWORD | size of private data area |
| 28h | DWORD | asynchronous event capabilities (see Table 11-13) |
| 2Ch | BYTE | highest path ID assigned |
| 2Dh | BYTE | SCSI device ID of initiator |
| 2Eh | 2 BYTEs | reserved |
| 30h | 16 BYTEs | SIM vendor ID |
| 40h | 16 BYTEs | HBA (host bus adaptor) vendor ID |
| 50h | 4 BYTEs | operating-system dependant usage |
| **---functions 00h,04h,11h,12h---** | | |
| no additional fields | | |
| **---function 05h---** | | |
| 10h | DWORD | asynchronous event enables (see function 03h above) |
| 14h | DWORD | pointer to asynchronous callback routine |
| 18h | DWORD | pointer to peripheral driver buffer |
| 1Ch | BYTE | size of peripheral buffer |
| **---function 06h---** | | |
| 10h | BYTE | peripheral device type of target |
| **---functions 10h,13h---** | | |
| 10h | DWORD | pointer to CCB to be aborted |
| **---function 20h---** | | |
| 10h | WORD | engine number |
| 12h | BYTE | engine type<br>00h buffer memory<br>01h lossless compression<br>02h lossy compression<br>03h encryption |
| 13h | BYTE | engine algorithm ID |

## INT 4Fh Function 8100h

| | | |
|---|---|---|
| | | 00h vendor-unique |
| | | 01h LZ1 variation 1 (STAC) |
| | | 02h LZ2 variation 1 (HP DCZL) |
| | | 03h LZ2 variation 2 (Infochip) |
| 14h | DWORD | engine memory size |
| **---function 21h---** | | |
| 10h | DWORD | pointer to peripheral driver |
| 14h | 4 BYTEs | reserved |
| 18h | DWORD | OS-dependent request-mapping info |
| 1Ch | DWORD | address of completion callback routine |
| 20h | DWORD | pointer to scatter/gather list or data buffer |
| 24h | DWORD | length of data transfer |
| 28h | DWORD | pointer to engine buffer data |
| 2Ch | 2 BYTEs | reserved |
| 2Eh | WORD | number of scatter/gather entries |
| 30h | DWORD | maximum destination data length |
| 34h | DWORD | length of destination data |
| 38h | DWORD | source residual length |
| 3Ch | 12 BYTEs | reserved |
| 48h | DWORD | OS-dependent timeout value |
| 4Ch | 4 BYTEs | reserved |
| 50h | WORD | engine number |
| 52h | WORD | vendor-unique flags |
| 54h | 4 BYTEs | reserved |
| 58h | N BYTEs | private data area for SIM |
| **---function 30h---** | | |
| 10h | WORD | group 6 vendor-unique CDB length |
| 12h | WORD | group 7 vendor-unique CDB length |
| 14h | DWORD | pointer to target CCB list |
| 18h | WORD | number of target CCBs |
| **---other functions---** | | |
| 10h | DWORD | pointer to peripheral driver |
| 14h | DWORD | pointer to next CCB |
| 18h | DWORD | OS-dependent request mapping information |
| 1Ch | DWORD | address of completion callback routine |
| 20h | DWORD | pointer to scatter/gather list or data buffer |
| 24h | DWORD | length of data transfer |
| 28h | DWORD | pointer to sense info buffer |
| 2Ch | BYTE | length of sense info buffer |
| 2Dh | BYTE | CDB length |
| 2Eh | WORD | number of scatter/gather entries |
| 30h | 4 BYTEs | reserved |
| 34h | BYTE | SCSI status |
| 35h | 3 BYTEs | reserved |
| 38h | DWORD | residual length |
| 40h | 12 BYTEs | Command Descriptor Block (CDB) |
| 44h | DWORD | OS-dependent timeout value |
| 48h | DWORD | pointer to message buffer |
| 4Ch | WORD | length of message buffer |

| | | |
|---|---|---|
| 4Eh | WORD | vendor-unique flags |
| 50h | BYTE | tag queue action |
| 51h | 3 BYTEs | reserved |
| 54h | N BYTEs | private data area for SIM |

### 11-8 Bitfields for CAM flags:

| | |
|---|---|
| bit 0 | CDB is a pointer |
| bit 1 | tagged queue action enable |
| bit 2 | linked CDB |
| bit 3 | disable callback on completion |
| bit 4 | scatter/gather |
| bit 5 | disable autosense |
| bits 7-6 | direction (00 reserved, 01 in, 10 out, 11 no data transfer) |
| bits 9-8 | reserved |
| bit 10 | engine synchronize |
| bit 11 | SIM queue freeze |
| bit 12 | SIM queue priority<br>1 head insertion<br>0 tail insertion (normal) |
| bit 13 | disable synchronous transfers |
| bit 14 | initiate synchronous transfers |
| bit 15 | disable disconnect |

**Note:** bits 13 and 14 are mutually exclusive

### 11-9 Bitfields for CAM address flags:

| | |
|---|---|
| bit 7 | SG list/data (0 = host, 1 = engine) |
| bit 6 | CDB pointer (bits 6-1: 0=virtual addr, 1=phys addr) |
| bit 5 | SG list/data |
| bit 4 | sense buffer |
| bit 3 | message buffer |
| bit 2 | next CCB |
| bit 1 | callback on completion |
| bit 0 | reserved |

### 11-10 CAM status values:

| | |
|---|---|
| 00h | request in progress |
| 01h | request successful |
| 02h | host aborted request |
| 03h | unable to abort request |
| 04h | request completed with error |
| 05h | CAM is busy |
| 06h | invalid request |
| 07h | invalid path ID |
| 08h | no such SCSI device |
| 09h | unable to terminate I/O process |
| 0Ah | timeout on target selection |
| 0Bh | timeout on command |
| 0Dh | receive message rejection |
| 0Eh | sent/received SCSI bus reset |
| 0Fh | detected uncorrectable parity error |
| 10h | Autosense request failed |
| 11h | no HBA detected |

## INT 4Fh Function 8100h

| | |
|---|---|
| 12h | data over/underrun |
| 13h | bus freed unexpectedly |
| 14h | target bus phase sequence failure |
| 15h | CCB too small |
| 16h | requested capability not available |
| 17h | sent bus device reset |
| 18h | terminate I/O process |
| 38h | invalid LUN |
| 39h | invalid target ID |
| 3Ah | unimplemented function |
| 3Bh | nexus not established |
| 3Ch | invalid initiator ID |
| 3Dh | received SCSI Command Descriptor Block |
| 3Eh | LUN already enabled |
| 3Fh | SCSI bus busy |

**Note:** Bit 6 is set to indicate a frozen SIM queue, bit 7 is set to indicate valid autosense.

**11-11 Bitfields for target-mode flags:**

| | |
|---|---|
| bit 7 | data buffer valid |
| bit 6 | status valid |
| bit 5 | message buffer valid |
| bit 4 | reserved |
| bit 3 | phase-cognizant mode |
| bit 2 | target CCB available |
| bit 1 | disable autodisconnect |
| bit 0 | disable autosave/restore |

**11-12 Bitfields for SCSI capabilities:**

| | |
|---|---|
| bit 7 | modify data pointers |
| bit 6 | wide bus (32 bits) |
| bit 5 | wide bus (16 bits) |
| bit 4 | synchronous transfers |
| bit 3 | linked commands |
| bit 2 | reserved |
| bit 1 | tagged queueing |
| bit 0 | soft reset |

**11-13 Bitfields for asynchronous event capabilities:**

| | |
|---|---|
| bits 31-24 | vendor-specific |
| bits 23-8 | reserved |
| bit 7 | new devices found during rescan |
| bit 6 | SIM module deregistered |
| bit 5 | SIM module registered |
| bit 4 | sent bus device reset to target |
| bit 3 | SCSI AEN |
| bit 2 | reserved |
| bit 1 | unsolicited reselection |
| bit 0 | unsolicited SCSI bus reset |

### 11-14 Completion callback function calling convention:

**Registers at call:**
interrupts disabled
ES:BX -> completed CCB

**Return Registers:**
all registers preserved

### 11-15 Asynchronous callback function calling convention:

**Registers at call:**
AH = opcode
AL = path ID generating callback
DH = target ID causing event
DL = LUN causing event
CX = data byte count (if applicable)
ES:BX -> data buffer (if applicable)

**Return Registers:**
all registers preserved

## INT 4Fh Function 8200h

*CAM revision 2.3*

### *INSTALLATION CHECK*

**Registers at call:**
AX = 8200h
CX = 8765h
DX = CBA9h

**Return Registers:**
AH = 00h if installed
CX = 9ABCh
DX = 5678h
ES:DI -> "SCSI_CAM"

**Conflicts:** Z100 (Chapter 7)
**See also:** Function 8100h, INT 4Bh

# CMC International TARGA.DEV

## INT 78h Function 00h

*TARGA.DEV*

### *CMC International SCSI device driver - SET I/O PORT*

**Registers at call:**
AH = 00h
DX = interface board I/O port

**Return Registers:**
CF set on error
AL = error code (see Table 11-16)

**Details:** If this routine is not called, the port is the driver's default (usually 0280h or 0300h).

An installation check is performed by TARGA.DEV upon initialization by checking for the string "SCSI" at offset 03h into the interrupt handler.

**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)
**See also:** Functions 01h and 02h

### 11-16 TARGA error codes:

| | |
|---|---|
| 00h | illegal command given to SCSI code |
| 01h | invalid I/O port specified (must be from 100H to 3F8H, and on an 8-port boundary) |
| 02h | invalid DMA channel specified (must be from 1 to 3) |
| 03h | invalid SCSI board number specified (must be from 0 to 7) |
| 04h | error from data register test during self-test |
| 05h | SCSI input signals not all 0 when SCSI RST activated |
| 06h | SCSI input signals not all 0 before selecting a SCSI device |
| 07h | BSY signal is active; SCSI bus is busy |
| 08h | SCSI board not selected, BSY signal did not come on in response to raising SEL |
| 09h | time-out waiting for status state, signifying end of DMA transfer |

---

**INT 78h Function 01h** *TARGA.DEV*

***GET I/O PORT***

**Registers at call:**
AH = 01h

**Return Registers:**
DX = current interface board I/O port

**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)

**See also:** Functions 00h and 03h

---

**INT 78h Function 02h** *TARGA.DEV*

***SET DMA CHANNEL***

**Registers at call:**
AH = 02h
AL = interface board DMA channel

**Return Registers:**
CF set on error
AL = error code (see Table 11-16)

**Details:** If this routine is not called, the DMA channel is the driver's default (usually 3).

**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)

**See also:** Functions 00h and 03h

---

**INT 78h Function 03h** *TARGA.DEV*

***GET DMA CHANNEL***

**Registers at call:**
AH = 03h

**Return Registers:**
AL = current interface board DMA channel

**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)

**See also:** Functions 01h and 02h

---

**INT 78h Function 04h** *TARGA.DEV*

***SET SCSI DEVICE NUMBER***

**Registers at call:**
AH = 04h
AL = SCSI device number

**Return Registers:**
CF set on error
AL = error code (see Table 11-16)

**Details:** If this routine is not called, the device number used is the driver's default (usually 0).

**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)

**See also:** Functions 02h and 05h

---

**INT 78h Function 05h** *TARGA.DEV*

***GET SCSI DEVICE NUMBER***

**Registers at call:**
AH = 05h

**Return Registers:**
AL = current SCSI device number

**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)

**See also:** Functions 03h and 04h

---

**INT 78h Function 06h** *TARGA.DEV*

***SET/CLEAR EARLY RETURN MODE***

**Registers at call:**
AH = 06h
AL = operation
    00h clear early return mode
    01h set early return mode

**Return Registers:**
none

**Details:** If early return mode is set then SCSI will return with no errors when the last DMA transfer is started in a call with Function 13h or 14h.

If this routine is not called, early return mode is cleared.
**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)
**See also:** Functions 13h, 14h, and 15h

---

### INT 78h Function 08h
*TARGA.DEV*

### *INTERFACE BOARD SELF-TEST*

**Registers at call:**
AH = 08h

**Return Registers:**
CF set on error
AL = error code (see Table 11-16)

**Details:** The SCSI bus is also reset.
**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)
**See also:** Function 09h

---

### INT 78h Function 09h
*TARGA.DEV*

### *RESET SCSI BUS*

**Registers at call:**
AH = 09h

**Return Registers:**
AL = error code if carry set (see Table 11-16)

**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)
**See also:** Function 08h

---

### INT 78h Function 10h
*TARGA.DEV*

### *SEND SCSI COMMAND*

**Registers at call:**
AH = 10h
DS:SI -> SCSI command (see Table 11-17)

**Return Registers:**
AH = SCSI status byte
CF clear if successful
  AL = SCSI message byte
CF set on error
  AL = error code (see Table 11-16)

**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)
**See also:** Function 11h, ST-01 INT 21h Function 4403h

**11-17 Format of SCSI Command:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | length of command |
| 01h | var | command bytes |

---

### INT 78h Function 11h
*TARGA.DEV*

### *SEND SCSI COMMAND, RECEIVE DATA (PROGRAMMED I/O)*

**Registers at call:**
AH = 11h
DS:SI -> SCSI command (see Table 11-17)
ES:BX -> data storage area
CX = number of data bytes to transfer

**Return Registers:**
AH = SCSI status byte
CF clear if successful
  AL = SCSI message byte
CF set on error
  AL = error code (see Table 11-16)

**Details:** This command receives data internally one byte at a time.
**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)
**See also:** Functions 10h and 13h

### INT 78h Function 12h

*TARGA.DEV*

### *SEND SCSI COMMAND AND DATA (PROGRAMMED I/O)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 12h | AH = SCSI status byte |
| DS:SI -> SCSI command (see Table 11-17) | CF clear if successful |
| ES:BX -> data storage area | AL = SCSI message byte |
| CX = number of data bytes to transfer | CF set on error |
| | AL = error code (see Table 11-16) |

**Details:** This command sends data internally one byte at a time.

**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)

**See also:** Function 14h

### INT 78h Function 13h

*TARGA.DEV*

### *SEND SCSI COMMAND, RECEIVE DATA (DMA)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 13h | AH = SCSI status byte (if early return mode is clear) |
| DS:SI -> SCSI command (see Table 11-17) | CF clear if successful |
| ES:BX -> data storage area | AL = SCSI message byte (if early return mode is clear) |
| DX:CX = number of data bytes to transfer | CF set on error |
| | AL = error code (see Table 11-16) |

**Details:** This command receives data using DMA.

**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)

**See also:** Functions 11h and 12h

### INT 78h Function 14h

*TARGA.DEV*

### *SEND SCSI COMMAND AND DATA (DMA)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 14h | AH = SCSI status byte (if early return mode is clear) |
| DS:SI -> SCSI command (see Table 11-17) | CF clear if successful |
| ES:BX -> data storage area | AL = SCSI message byte (if early return mode is clear) |
| DX:CX = number of data bytes to transfer | CF set on error |
| | AL = error code (see Table 11-16) |

**Details:** This command sends data using DMA.

**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)

**See also:** Functions 12h and 13h

### INT 78h Function 15h

*TARGA.DEV*

### *FINISH DATA TRANSFER (DMA)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 15h | AH = SCSI status byte |
| | CF clear if successful |
| | AL = SCSI message byte |
| | CF set on error |
| | AL = error code (see Table 11-16) |

**Details:** If Function 06h was previously called to set the early return mode, this function finishes a command Functions 13h or 14h which returned before the last DMA transfer was finished.

**Conflicts:** ADP-60 (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD ADI (Chapter 66)
**See also:** Functions 06h, 13h, and 14h

## Future Domain

### INT 13h Function 05h

*Future Domain SCSI BIOS*

#### *SEND SCSI MODE SELECT COMMAND*

**Registers at call:**
AH = 05h
DL = hard drive ID
ES:BX -> mode select data (see Table 11-18)

**Return Registers:**
CF set on error
AH = status code (see Table 10-1)

**Details:** This function can be called before Functions 06h or 07h to format a SCSI disk with the desired parameters.

The mode select data below is from the SCSI-1 specification.

The TMC-950 does not support any Future Domain BIOS calls; instead, it provides a full CAM implementation (see INT 4Fh Function 8100h).

**Conflicts:** Standard INT 13h (Chapter 10)
**See also:** SCSI Functions 06h and 07h, INT 4Fh Function 8100h (above)

**11-18 Format of mode select data:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | number of bytes of remaining data (12 + vendor unique length) |
| 01h | BYTE | reserved (0) |
| 02h | BYTE | medium type (0 for hard disk) |
| 03h | BYTE | reserved (0) |
| 04h | BYTE | block descriptor length (8) |
| 05h | BYTE | density code (0 for hard disk) |
| 06h | 3 BYTEs | (big-endian) number of blocks (000000h for entire disk) |
| 09h | BYTE | reserved (0) |
| 0Ah | 3 BYTEs | (big-endian) block length (512 standard, or 256) |
| 0Dh | var | vendor-specific parameter bytes (optional) |

### INT 13h Function 06h

*Future Domain SCSI BIOS*

#### *FORMAT DRIVE WITH BAD SECTOR MAPPING*

**Registers at call:**
AH = 06h
AL = interleave
  (0 = default, 1 = consecutive sectors,
   2 - 255 = vendor unique)
DL = hard drive ID
DH = bits 7-5 drive LUN
  bit 4 defect list is available
  bit 3 defect list is complete (erase drive's defect list)
  bits 2-0 defect table format
    (000=use defect table A, 100=use defect table B, 101=use defect table C)
ES:BX -> defect table A, B or C (see Tables 11-19 to 11-21)

**Return Registers:**
CF set on error
AH = status code (see Table 10-1)

**Details:** Block addresses must be in ascending order (for table B, cylinder is most significant, byte from index least significant; for table C, cylinder is most significant, sector number least significant).

Table B defect bytes from index of FFFFFFFFh indicates that the entire track shall be reassigned.

Table C defect sector number of FFFFFFFFh indicates that the entire track shall be reassigned.

The TMC-950 does not support any Future Domain BIOS calls; instead, it provides a full CAM implementation (see INT 4Fh Function 8100h).

**Conflicts:** Standard INT 13h (Chapter 10), Adaptec AHA-154xA (below)

**See also:** SCSI Functions 05h and 07h, Fixed Disk Function 06h (Chapter 10)

### 11-19 Format of defect table A:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of bytes remaining in table |
| 02h | BYTE | reserved (0) |
| 03h | BYTE | reserved (0) |
| 04h | WORD | (big-endian) defect list length (4*number of defects) |
| 06h | 4 DWORDs | (big-endian) defect block addresses |

### 11-20 Format of defect table B:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of bytes remaining in table |
| 02h | BYTE | reserved (0) |
| 03h | BYTE | reserved (0) |
| 04h | WORD | (big-endian) defect list length (8*number of defects) |
| 06h | 8N BYTEs | defect list [array] (see Table 11-22) |

### 11-21 Format of defect table C:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of bytes remaining in table |
| 02h | BYTE | reserved (0) |
| 03h | BYTE | reserved (0) |
| 04h | WORD | (big-endian) defect list length (8*number of defects) |
| 06h | 8N BYTEs | defect list [array] (see Table 11-22) |

### 11-22 Format of defect list entry:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 3 BYTEs | (big-endian) cylinder number of defect |
| 03h | BYTE | head number of defect |
| 04h | DWORD | (big-endian) defect bytes from index |

---

## INT 13h Function 07h

*Future Domain SCSI BIOS*

### *FORMAT DRIVE*

**Registers at call:**

AH = 07h

AL = interleave (0 = default, 1 = consecutive sectors, 2 - 255 = vendor unique)

DL = hard drive ID

**Return Registers:**

CF set on error

AH = status code (see Table 10-1)

**Conflicts:** Standard INT 13h (Chapter 10)

**See also:** SCSI Functions 05h and 06h, Fixed Disk Function 07h (Chapter 10)

---

## INT 13h Function 12h

*Future Domain SCSI CONTROLLER*

### *STOP SCSI DISK*

**Registers at call:**

AH = 12h

DL = hard drive ID

**Return Registers:**

CF set on error

AH = status code (see Table 10-1)

**Details:** This function is available on at least the TMC-870 8-bit SCSI controller BIOS v6.0A.

If the given drive is a SCSI device, the SCSI Stop Unit command is sent and either "Disk prepared for shipping" or "Disk Stop command failed" is displayed.

The TMC-950 does not support any Future Domain BIOS calls; instead, it provides a full CAM implementation (see INT 4Fh Function 8100h).

---

## INT 13h Function 1700h

*Future Domain SCSI CONTROLLER*

### *GET INQUIRY INFO FROM SCSI DEVICE*

**Registers at call:**
AX = 1700h
CL = length of buffer
DL = hard drive ID
ES:BX -> buffer for info (see Table 11-23)

**Return Registers:**
CF clear if successful
  CH = *number of bytes returned in buffer*
CF set on error
  AH = status code (see Table 10-1)

**Details:** This function is not available with 8-bit controller ROM versions prior to 7.0.

Information block bytes 5-n are vendor-specific in older SCSI devices.

The TMC-950 does not support any Future Domain BIOS calls; instead, it provides a full CAM implementation (see INT 4Fh Function 8100h).

**Conflicts:** Standard INT 13h (Chapter 10)

**See also:** SCSI Functions 18h and 1Bh

**11-23 Format of SCSI inquiry information block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | device type<br>bits 0-4: peripheral device type (see Table 11-24)<br>bits 5-7: peripheral qualifier (see Table 11-25) |
| 01h | BYTE | device type modifier<br>bits 0-6: device type modifier<br>bit 7: removable medium |
| 02h | BYTE | SCSI version<br>bits 0-2: ANSI-approved version<br>000 device might or might not comply to ANSI standard<br>001 device complies to ANSI SCSI-1<br>010 device complies to ANSI SCSI-2<br>other reserved<br>bits 3-5: ECMA version<br>bits 6,7: ISO version |
| 03h | BYTE | data format/capabilities<br>bits 0-2: response data format<br>000 information block is as specified in SCSI-1<br>001 information block is as specified in CCS<br>010 information block is as specified in SCSI-2<br>other reserved<br>bits 4,5: reserved<br>bit 6: terminate I/O process supported<br>bit 7: asynchronous event notification supported |
| 04h | BYTE | additional data length (total remaining bytes) |
| 05h | 2 BYTEs | reserved |
| 07h | BYTE | device capabilities (see Table 11-26) |
| 08h | 8 BYTEs | vendor identification (space-padded ASCII) |
| 10h | 8 BYTEs | product identification (space-padded ASCII) |
| 20h | 4 BYTEs | product revision level (space-padded ASCII) |
| 24h | 20 BYTEs | vendor specific |

| | | |
|---|---|---|
| 38h | 40 BYTEs | reserved |
| 60h | var | vendor specific parameters |

**11-24 Peripheral device types:**

| | |
|---|---|
| 00h | direct-access device (e.g., magnetic disk) |
| 01h | sequential-access device (e.g., magnetic tape) |
| 02h | printer device |
| 03h | processor device |
| 04h | write-once device (e.g., some optical disks) |
| 05h | CD-ROM device |
| 06h | scanner device |
| 07h | optical memory device (e.g., some optical disks) |
| 08h | medium changer device (e.g., jukeboxes) |
| 09h | communications device |
| 0Ah | (defined by ASC IT8) |
| 0Bh | (defined by ASC IT8) |
| 0Ch-1Eh | reserved |
| 1Fh | unknown or no device type |

**11-25 Peripheral qualifiers:**

| | |
|---|---|
| 000b | device is currently connected to this logical unit and available |
| 001b | target is capable of supporting the specified peripheral, but the physical device is not currently connected to this logical unit |
| 010b | reserved |
| 011b | target can't support a physical device on this logical unit |
| 1xxb | vendor specific |

**11-26 Bitfields for device capabilities:**

| | |
|---|---|
| bit 0 | device responds to RESET with a hard RESET |
| bit 1 | tagged command queuing supported |
| bit 2 | reserved |
| bit 3 | linked commands supported |
| bit 4 | synchronous data transfer supported |
| bit 5 | 16-bit transfers supported |
| bit 6 | 32-bit transfers supported |
| bit 7 | relative addressing supported |

---

## INT 13h Function 18h
*Future Domain SCSI BIOS*

### *GET SCSI CONTROLLER INFORMATION*

**Registers at call:**
AH = 18h
DL = hard drive ID

**Return Registers:**
CF set on error
AH = status code (see Table 10-1)
CF clear if successful
AX = 4321h (magic number)
CX = controller family code (see Table 11-27)
**if family code=0200h**
DH = number of exclusively ROM-controlled SCSI devices
DL = canonical SCSI device number for specified drive
**if family code ≠ 0200h**
BH = number of exclusively ROM-controlled SCSI devices

BL = canonical SCSI device number for specified drive

**Details:** Also sets an internal flag (non-resettable) which prevents some controller messages from being displayed, allows writes to removable devices (use caution!), and enables the INT 13h interface for more than one drive (i.e. DL >= 81h) in at least some ROM versions.

The TMC-950 does not support any Future Domain BIOS calls; instead, it provides a full CAM implementation (see INT 4Fh Function 8100h).

**Conflicts:** Standard INT 13h (Chapter 10)

**See also:** SCSI Functions 05h, 1700h, and 1Bh, INT 4Fh Function 8100h (above)

**11-27 Controller family codes:**

| | |
|---|---|
| 0200h | TMC-1680/? (ROM 3.0) |
| 0203h | TMC-1650/1660/1670/1680 (ROM 2.0) |
| 040Ah | TMC-820/830/840/850/860/870/875/880/885 (ROM <= 6.0A) |
| 050Dh | TMC-840/841/880/881 (ROM 5.2D) |
| 0700h | TMC-830/850/860/875/885 (ROM 7.0) |

## INT 13h Function 19h

*Future Domain SCSI CONTROLLER*

### *REINITIALIZE DRIVE*

**Registers at call:**

AH = 19h
DL = hard drive ID

**Return Registers:**

CF set on error
AH = status code (see Table 10-1)
CF clear if successful
AH = disk type (03h = fixed disk)
CX:DX = number of 512-byte sectors

**Details:** Sends SCSI Read Capacity command to get number of logical blocks and adjusts the result for 512-byte sectors.

Displays either "Error in Read Capacity Command" or "nnn Bytes per sector" (nnn=256 or 512, the only sizes supported in the translation code).

Should probably be called when a removable device has its media changed.

Returns the same values as Function 15h.

The TMC-950 does not support any Future Domain BIOS calls; instead, it provides a full CAM implementation (see INT 4Fh Function 8100h).

**Conflicts:** Standard INT 13h (Chapter 10)

**See also:** CAM INT 4Fh Function 8100h (below), Standard Function 15h (Chapter 10), ESDI Function 1Ah (Chapter 10)

## INT 13h Function 1Ah

*Future Domain SCSI CONTROLLER*

### *GET SCSI PARTIAL MEDIUM CAPACITY*

**Registers at call:**

AH = 1Ah
CH = track (bits 8,9 in high bits of CL)
CL = sector (01h to number of sectors/track for drive)
DH = head
DL = hard drive ID

**Return Registers:**

CF set on error
AH = status code (see Table 10-1)
CX:DX = logical block number of last quickly-accessible block after given block

**Details:** This function sends a SCSI Read Capacity command with the PMI bit set to obtain the logical block address of the last block after which a substantial delay in data transfer will be encountered (usually the last block on the current cylinder). No translation to 512-byte sectors is performed on the result if data is stored on the disk in other than 512-byte sectors.

The TMC-950 does not support any Future Domain BIOS calls; instead, it provides a full CAM implementation (see INT 4Fh Function 8100h below).

**Conflicts:** ESDI Fixed Disk (Chapter 10)

**See also:** Function 19h, Standard Function 15h (Chapter 10)

## INT 13h Function 1Bh

*Future Domain SCSI CONTROLLER*

### GET POINTER TO SCSI DISK INFO BLOCK

**Registers at call:**
AH = 1Bh
DL = hard drive ID

**Return Registers:**
CF set on error
  AH = status code (see Table 10-1)
CF clear if successful
  ES:BX -> SCSI disk information block (see Table 11-28)

**Details:** This function also sets a non-resettable flag which prevents some controller messages from being displayed. The TMC-950 does not support any Future Domain BIOS calls; instead, it provides a full CAM implementation (see INT 4Fh Function 8100h).

**Conflicts:** ESDI Fixed Disk (Chapter 10)

**See also:** Functions 18h and 1Ch

**11-28 Format of SCSI disk information block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | drive physical information (see Table 11-29) |
| 01h | WORD | translated number of cylinders |
| 03h | BYTE | translated number of heads |
| 04h | BYTE | translated number of sectors per track (17, 34, or 63) |
| 05h | BYTE | drive address<br>bits 0-2: logical unit number<br>bits 3-5: device number |
| 06h | BYTE | 01h at initialization |
| 07h | BYTE | sense code byte 00h, or extended sense code byte 0Ch |
| 08h | BYTE | 00h |
| 09h | BYTE | 00h or extended sense code byte 02h (sense key) |
| 0Ah | BYTE | 00h |
| 0Bh | 10 BYTEs | copy of Command Descriptor Block (CDB) |
| 15h | DWORD | translated number of sectors on device |

**11-29 Bitfields for device physical information:**

| | |
|---|---|
| bit 0 | *unknown* |
| bit 1 | device uses parity |
| bit 2 | 256 bytes per sector instead of 512 |
| bit 3 | *don't have capacity yet* |
| bit 4 | disk is removable |
| bit 5 | logical unit number is not present |

## INT 13h Function 1Ch

*Future Domain SCSI CONTROLLER*

### GET POINTER TO FREE CONTROLLER RAM

**Registers at call:**
AH = 1Ch
DL = hard drive ID (any valid SCSI hard disk)

**Return Registers:**
CF set on error
  AH = status code (see Table 10-1)
CF clear if successful
  ES:BX -> first byte of free RAM on controller

**Details:** The Future Domain TMC-870 contains 1024 bytes of RAM at offsets 1800h to 1BFFh on-board the controller for storing drive information and controller status; ES:BX points to the first byte available for other uses.

ES contains the segment at which the controller resides; the controller's two memory-mapped I/O ports are at offsets 1C00h and 1E00h.

**Conflicts:** ESDI Fixed Disk (Chapter 10)

**See also:** Function 1Bh

## SCSILink

SCSILink is a TSR by Cross Products, Ltd which allows its PC-hosted cross assemblers and similar products to communicate with Cross Products hardware debuggers.

---

### INT 7Dh Function 00h

*SCSILink*

***RESET SCSI BUS***

**Purpose:** Reset all devices on the SCSI bus.

**Registers at call:**
AH = 00h

**Return Registers:**
none

**Details:** The installation check is the signature "SCSILINK" immediately prior to the interrupt handler.

**Conflicts:** G032 (Chapter 37), HyperPAD (Chapter 66), YTERM (Chapter 66)

---

### INT 7Dh Function 01h

*SCSILink*

***CONNECT TO TARGET***

**Purpose:** Arbitrate the use of the bus and select the specified target device.

**Registers at call:**
AH = 01h
AL = target ID

**Return Registers:**
CF clear if successful
CF set on error
  AL = initiator error
  AH = target error

**Conflicts:** G032 (Chapter 37), HyperPAD (Chapter 66), YTERM (Chapter 66)

---

### INT 7Dh Function 02h

*SCSILink*

***SEND COMMAND***

**Purpose:** Send the specified command block to the target device and perform any related I/O.

**Registers at call:**
AH = 02h
ES:BX -> parameter block (see Table 11-30)

**Return Registers:**
CF clear if successful
CF set on error
  AL = initiator error
  AH = target error

**Conflicts:** G032 (Chapter 37), HyperPAD (Chapter 66), YTERM (Chapter 66)

**11-30 Format of parameter block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | size of command block |
| 04h | DWORD | address of command block (see Table 11-31) |
| 08h | DWORD | size of buffer |
| 0Ch | DWORD | address of buffer |

**11-31 Format of command block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | | info not yet available |

---

### INT 7Dh Function 03h

*SCSILink*

***SET TIMEOUT***

**Purpose:** Change an internal timeout value to allow communication with very slow targets.

**Registers at call:**
AH = 03h
AL = timeout selector (see Table 11-32)
BX = new value in 55ms ticks

**Return Registers:**
none

**Conflicts:** G032 (Chapter 37), HyperPAD (Chapter 66), YTERM (Chapter 66)

**11-32 Values for timeout selector:**

| | |
|---|---|
| 00h | time to wait for bus (default 18) |
| 01h | time to wait for new phase (default 5) |
| 02h | maximum time to send/receive block (default 18) |
| 03h | time to wait for reselect (default 180) |

---

### INT 7Dh Function 04h

*SCSILink*

***GET ERROR STRING***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 04h | ES:BX -> ASCIZ error string for error number |
| AL = error number | |

**Conflicts:** G032 (Chapter 37), HyperPAD (Chapter 66), YTERM (Chapter 66)

---

### INT 7Dh Function 05h

*SCSILink*

***GET ADDRESS***

**Purpose:** Determine the current hardware configuration.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 05h | AL = DMA channnel |
| | AH = initiator ID |
| | BX = card address |

**Conflicts:** G032 (Chapter 37), HyperPAD (Chapter 66), YTERM (Chapter 66)

---

### INT 7Dh Function 06h

*SCSILink*

***PUT DATA***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 06h | CF clear if successful |
| CX = number of bytes to store (0001h-0100h) | CF set on error |
| ES:BX -> data to be saved | AL = error code (01h = too much data) |

**Details:** The specified data is stored in SCSILink's PSP.

**Conflicts:** G032 (Chapter 37), HyperPAD (Chapter 66), YTERM (Chapter 66)

**See also:** Function 07h

---

### INT 7Dh Function 07h

*SCSILink*

***GET DATA***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 07h | CF clear if successful |
| CX = number of bytes to retrieve (0001h-0100h) | CF set on error |
| ES:BX -> buffer for data | AL = error code (01h = too much data) |

**Details:** This function retrieves data previously stored with Function 06h.

**Conflicts:** G032 (Chapter 37), HyperPAD (Chapter 66), YTERM (Chapter 66)

**See also:** Function 06h

---

### INT 7Dh Function 08h

*SCSILink*

***TERMINATE SESSION***

**Purpose:** Indicate to any other programs that intercept INT 7Dh that the program has finished with the link.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 08h | none |

**Details:** The program should call this function even if a SCSI error caused its termination.

**Conflicts:** G032 (Chapter 37), HyperPAD (Chapter 66), YTERM (Chapter 66)

### INT 7Dh Function 09h

SCSILink

### *POLL REQUEST*

**Purpose:** Give any drivers chained onto INT 7Dh a chance to handle an exception not specifically handled by the calling program.

**Registers at call:**
AH = 09h
AL = target ID

**Return Registers:**
AL = status
  FEh resident driver experienced SCSI error
  FFh resident driver handleded event

**Details:** This service exists so that resident disk servers, etc. can continue running even while debuggers and profilers are active.

The resident driver assumes that a connection has been established and attempts to leave the target connected.

**Conflicts:** G032 (Chapter 37), HyperPAD (Chapter 66), YTERM (Chapter 66)

## SDLP

### INT 11h Function 6A6Ah

Columbia Data Products

### *Standard Device Level Protocol (SDLP) 1.6*

**Registers at call:**
SI = 6A6Ah
AH = command (see Table 11-33)
AL = SCSI Addressing
  bits 2-0 SCSI Target LUN (logical unit number)
  bits 5-3 SCSI Target ID
  bits 7-6 Host Adapter

**Return Registers:**
CF clear if successful
  DI = 6A6Ah if AH=01h on entry (*possibly for all functions*)
  AH = *unknown value* for command 01h
CF set on error
  AL = error code

**Conflicts:** Alignment Check (Chapter 3), Standard INT 11h (Chapter 4), BNU FOSSIL (Chapter 12), Blank-It (Chapter 14), Back and Forth (Chapter 49), Borland C++ IDE (Chapter 54), RainboxFAX (Chapter 64)

**See also:** ASPI INT 21h Function 4402h (above)

**11-33 SDLP commands:**

| | |
|---|---|
| 00h | SDLP initialization |
| 01h | SDLP System Identify |
| 02h | simple read sectors |
| 03h | simple write sectors |
| 04h | simple verify sectors/seek to sector |
| 05h | get device size/type |
| 06h | ready unit |
| 07h | format unit |
| 08h | diagnostics |
| 09h | rewind |
| 0Ah | erase |
| 0Bh | write filemarks |
| 0Ch | space |
| 0Dh | prevent/allow media removal |
| 0Eh | load/unload media |
| 0Fh | reserved - returns good status |
| 10h | set block size |
| 11h | write setmark |
| 12h | set error level |

| | |
|---|---|
| 13h | get address of Request Sense Buffer |
| 14h | get SDLP error via Request Sense |
| F0h | Vendor Unique Function (WD7000-FASST2 only) |
| FDh | reset current SCSI HAC |
| FEh | get/set current SCSI HAC |
| FFh | execute SCSI command |

---

### INT 11h Function FFh
*WD7000 SDLP interface*

#### ***EXECUTE GENERIC SCSI COMMAND***

**Registers at call:**
AH = FFh
SI = 6A6Ah
AL = SCSI Addressing
  bits 2-0 SCSI Target LUN (logical unit number)
  bits 5-3 SCSI Target ID
  bit 7 write flag, set for write operations, clear otherwise
CX = bytes of data to be transmitted (max FFF0h)
DH = 00h
DL = length of SCSI Command Descriptor Block
DS:DI -> SCSI Command Descriptor Block
ES:BX -> data buffer

**Return Registers:**
CF set on error
  AL = error code
CF clear if successful

**Details:** Because of busmaster operations with WD7000FASST, avoid accessing video memory directly; check 386 memory manager for VDS support. The WD7000XTAT works with programmed I/O and does not have this limitation.
**Conflicts:** Alignment Check (Chapter 3), Standard INT 11h (Chapter 4), BNU FOSSIL (Chapter 12), Blank-It (Chapter 14), Back and Forth (Chapter 49), Borland C++ IDE (Chapter 54), RainboxFAX (Chapter 64)
**See also:** ASPI INT 21h Function 4402h (above), CD-ROM redirector INT 2Fh Function 7F01h (below)

## Miscellaneous

---

### INT 13h Function 06h
*Adaptec AHA-154xA/Bustek BT-542 BIOS*

#### ***IDENTIFY SCSI DEVICES***

**Purpose:** Determine the number of the first supported SCSI drive.

**Registers at call:**
AH = 06h

**Return Registers:**
AH = status code (see Table 10-1)
CF clear if successful
  AL = first drive supported
    (80h nonconcurrent operation, 81h concurrent operation)
CF set on error

**Details:** The return value is 80h when two SCSI drives are supported, 81h if only one SCSI drive is installed.
**Conflicts:** Future Domain SCSI (above), Standard INT 13h (Chapter 10)
**See also:** Function 08h

---

### INT 21h Function 4403h
*Brian Antoine Seagate ST-01 SCSI.SYS*

#### ***IOCTL - EXECUTE COMMANDS***

**Registers at call:**
AX = 4403h
BX = handle for device "SCSITAPE"
CX = number of bytes to write

DS:DX -> SCSITAPE control block
(see Table 11-34)

**Return Registers:**
CF clear if successful
AX = number of bytes actually written
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**See also:** ST-01 Function 4405h, TARGA INT 78h Function 10h (above)

**11-34 Format of SCSITAPE control block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | command type |
| | | 'F' Format (argument 1 = interleave, argument 2 = format type) |
| | | 'E' Erase |
| | | 'R' Rewind |
| | | 'L' Load |
| | | 'N' No Load |
| | | 'S' Space (argument 1 = count, argument 2 = type) |
| | | 'M' File Mark (argument 1 = count) |
| | | 'A' Reassign |
| 02h | WORD | argument 1 |
| 04h | WORD | argument 2 |
| 06h | WORD | segment of command buffer |
| 08h | WORD | offset of command buffer |
| 0Ah | WORD | length of command buffer |

## INT 21h Function 4405h

*Brian Antoine Seagate ST-01 SCSI.SYS*

### *IOCTL - EXECUTE COMMANDS*

**Registers at call:**
AX = 4405h
BX = drive number (00h = default, 01h = A:, etc)
CX = number of bytes to write
DS:DX -> SCSIDISK control block
(see ST-01 Function 4403h)

**Return Registers:**
CF clear if successful
AX = number of bytes actually written
CF set on error
AX = error code (01h,05h,06h,0Dh)
(see Table 24-2)

**Conflicts:** xDISK (Chapter 10), DoubleSpace (Chapter 16), COMBI-disk (Chapter 17)
**See also:** ST-01 Function 4403h

## INT 2Fh Function 7F00h

*CD-ROM redirector SCSI driver*

### *INSTALLATION CHECK*

**Registers at call:**
AX = 7F00h

**Return Registers:**
AL = FFh if installed

**Program:** This program by Jim Harper allows a CD-ROM to appear as a network drive.
**Conflicts:** none, but see Table 2-2
**See also:** Functions 7F01h, 7F02h, and 7F03h

## INT 2Fh Function 7F01h

*CD-ROM redirector SCSI driver*

### *DO COMMAND*

**Registers at call:**
AX = 7F01h
DS:DX -> command record (see Table 11-35)

**Return Registers:**
AL = status
00h successful
else error code

**Program:** This program by Jim Harper allows a CD-ROM to appear as a network drive.
**Conflicts:** none, but see Table 2-2

**See also:** Functions 7F00h and 7F02h, SDLP INT 11h Function FFh (above), ASPI INT 21h Function 4402h (above), CAM INT 4Fh Function 8100h (above)

**11-35 Format of command record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | ID |
| 01h | 10 BYTEs | CDB (Command Descriptor Block) for operation |
| 0Bh | WORD | segment of buffer |
| 0Dh | WORD | offset of buffer |
| 0Fh | BYTE | status |
| 10h | BYTE | sense |
| 12h | WORD | count |

---

## INT 2Fh Function 7F02h

*CD-ROM redirector SCSI driver*

### *DO RESET*

**Registers at call:**
AX = 7F02h

**Return Registers:**
none

**Program:** This program by Jim Harper allows a CD-ROM to appear as a network drive.
**Conflicts:** none, but see Table 2-2
**See also:** Functions 7F00h and 7F01h

---

## INT 2Fh Function 7F03h

*CD-ROM redirector SCSI driver*

### *UNINSTALL*

**Registers at call:**
AX = 7F03h

**Return Registers:**
AL = status
00h successful
01h unable to uninstall

**Program:** This program by Jim Harper allows a CD-ROM to appear as a network drive.
**Conflicts:** none, but see Table 2-2
**See also:** Function 7F00h

---

## INT 4Bh Function 80h

*IBM SCSI interface*

**Registers at call:**
AH = 80h

**Return Registers:**
*unknown*

*details not yet available*

**Conflicts:** Common Access Method (above), Z100 (Chapter 7), Acorn BBC Master (Chapter 7)

## CHAPTER 12

# Serial I/O

## Standard BIOS

### INT 14h Function 00h

SERIAL

***INITIALIZE PORT***

**Registers at call:**

AH = 00h
AL = port parameters
bits 7-5 data rate (110,150,300,600, 1200,2400,4800,9600 bps)
bits 4-3 parity (00 or 10 = none, 01 = odd, 11 = even)
bit 2 stop bits (set = 2, clear = 1)
bits 1-0 data bits (00 = 5, 01 = 6, 10 = 7, 11 = 8)
DX = port number (00h-03h) (04h-43h for Digiboard XAPCM232.SYS)

**Return Registers:**

AH = line status (see Table 12-1)
FFh if error on Digiboard XAPCM232.SYS
AL = modem status (see Table 12-2)

**Details:** The default handler is at F000h:E739h in IBM PC and 100% compatible BIOSes.

Since the PCjr supports a maximum of 4800 bps, attempting to set 9600 bps will result in 4800 bps.

Various network and serial-port drivers support the standard BIOS functions with interrupt-driven I/O instead of the BIOS's polled I/O.

**Conflicts:** FOSSIL (below), MMBIOS (below)

**See also:** Functions 04h and 05h, COMM-DRV Function 81h (below), COURIERS Functions 82h and 8Ch (below), MultiDOS Function 04h (Chapter 45), ARTICOM Function 8000h (NI)

### INT 14h Function 01h

SERIAL

***WRITE CHARACTER TO PORT***

**Registers at call:**

AH = 01h
AL = character to write
DX = port number (00h-03h)
(04h-43h for Digiboard XAPCM232.SYS)

**Return Registers:**

AH bit 7 clear if successful
AH bit 7 set on error
AH bits 6-0 = port status (see Table 12-1)

**Details:** Various network and serial-port drivers support the standard BIOS functions with interrupt-driven I/O instead of the BIOS's polled I/O.

**See also:** Function 02h, FOSSIL Function 0Bh (below), COURIERS Function 89h (below), ARTICOM Function 8000h (NI)

### INT 14h Function 02h

SERIAL

***READ CHARACTER FROM PORT***

**Registers at call:**

AH = 02h

AL = 00h (ArtiCom)
DX = port number (00h-03h (04h-43h for Digiboard XAPCM232.SYS))

**Return Registers:**
AH = line status (see Table 12-1)
AL = received character if AH bit 7 clear

**Details:** Will timeout if DSR is not asserted, even if function 03h returns data ready.

Various network and serial-port drivers support the standard BIOS functions with interrupt-driven I/O instead of the BIOS's polled I/O.

**Conflicts:** FOSSIL (below)

**See also:** Function 01h, FOSSIL Function 02h (below), COURIERS Function 84h (below), EBIOS Function FCh (below)

---

### INT 14h Function 03h

*SERIAL*

### *GET PORT STATUS*

**Registers at call:**
AH = 03h
AL = 00h (ArtiCom)
DX = port number (00h-03h) (04h-43h for Digiboard XAPCM232.SYS)

**Return Registers:**
AH = line status (see Table 12-1)
AL = modem status (see Table 12-2)
AX = 9E00h if disconnected (ArtiCom)

**See also:** Function 00h, COURIERS Function 81h (below), EBIOS Function FD02h (below), MultiDOS Function 07h (Chapter 45), ARTICOM Function 8000h (NI)

**12-1 Bitfields for line status:**

| | |
|---|---|
| bit 7 | timeout |
| bit 6 | transmit shift register empty |
| bit 5 | transmit holding register empty |
| bit 4 | break detected |
| bit 3 | framing error |
| bit 2 | parity error |
| bit 1 | overrun error |
| bit 0 | receive data ready |

**12-2 Bitfields for modem status:**

| | |
|---|---|
| bit 7 | carrier detect |
| bit 6 | ring indicator |
| bit 5 | data set ready |
| bit 4 | clear to send |
| bit 3 | delta carrier detect |
| bit 2 | trailing edge of ring indicator |
| bit 1 | delta data set ready |
| bit 0 | delta clear to send |

---

### INT 14h Function 04h

*SERIAL*

### *EXTENDED INITIALIZE (CONVERTIBLE,PS)*

**Registers at call:**
AH = 04h
AL = break status
  00h if break
  01h if no break
BH = parity (see Table 12-3)
BL = number of stop bits
  00h one stop bit
  01h two stop bits (1.5 if 5 bit word length)
CH = word length (see Table 12-4)

CL = bps rate (see Table 12-5)
DX = port number

**Return Registers:**
AX = port status code
(see Tables 12-1 and 12-2)

**Conflicts:** FOSSIL (below), MBBIOS (below), Digiboard (below), TSRCOMM (below), MultiDOS Plus (Chapter 45)
**See also:** Function 00h, FOSSIL Function 1Eh (below), ARTICOM Function 8000h (NI)

**12-3 Values for parity:**

| | |
|---|---|
| 00h | no parity |
| 01h | odd parity |
| 02h | even parity |
| 03h | stick parity odd |
| 04h | stick parity even |

**12-4 Values for word length:**

| | |
|---|---|
| 00h | 5 bits |
| 01h | 6 bits |
| 02h | 7 bits |
| 03h | 8 bits |

**12-5 Values for bps rate:**

| | |
|---|---|
| 00h | 110 |
| 01h | 150 |
| 02h | 300 |
| 03h | 600 |
| 04h | 1200 |
| 05h | 2400 |
| 06h | 4800 |
| 07h | 9600 |
| 08h | 19200 |

---

## INT 14h Function 05h

*SERIAL*

### *EXTENDED COMMUNICATION PORT CONTROL (CONVERTIBLE,PS)*

**Registers at call:**
AH = 05h
DX = port number
AL = function
00h read modem control register
01h write modem control register
BL = modem control register (see Table 12-6)

**Return Registers:**
(00h) AH = status
BL = modem control register
(see Table 12-6)
(01h) AX = status

**Details:** Also supported by ArtiCom.
**Conflicts:** FOSSIL (below), MBBIOS (below), Digiboard (below), MultiDOS Plus (Chapter 45)
**See also:** Function 00h, FOSSIL Function 1Fh (below), EBIOS Function FBh (below), ARTICOM Function 8000h (NI)

**12-6 Bitfields for modem control register:**

| | |
|---|---|
| bit 0 | data terminal ready |
| bit 1 | request to send |
| bit 2 | OUT1 |
| bit 3 | OUT2 |
| bit 4 | LOOP |
| bits 5-7 | reserved |

## Digiboard

This section covers the Digiboard DigiCHANNEL PC/X* Extender INT 14h (XAPCM232.SYS). The Digiboard is a multi-port serial adapter card which allows a single PC to support up to 64 serial ports.

The installation check for the XAPCM232.SYS driver is to determine whether the "-DOSXAM" character device exists.

---

### INT 14h Function 04h

*Digiboard DigiCHANNEL PC/X**

### *CHANGE BAUD RATE*

**Registers at call:**

AH = 04h
AL = initializing parameters
        bits 7-5 unused
        bits 4,3: parity (00 none, 01 odd, 11 even)
        bit 2: stop bits (0 = one, 1 = two)
        bits 1,0: data bits (00 = five, 01 = six, 10 = seven, 11 = eight)
BX = baud rate
DX = port number (00h-03h) (04h-43h for XAPCM232.SYS)

**Return Registers:**

AH = status
        00h successful
        FFh error

**Conflicts:** Standard Function 04h (above), FOSSIL (below), MBBIOS (below), TSRCOMM (below), MultiDOS Plus (Chapter 45)

**See also:** Digiboard Function 05h

---

### INT 14h Function 05h

*Digiboard DigiCHANNEL PC/X**

### *CHANGE PROTOCOL*

**Registers at call:**

AH = 05h
AL = protocol
        bits 7-4 unused
        bit 3: RTS/CTS
        bit 2: DSR
        bits 1,0: XON/XOFF
BH = new XOFF character (00h = current)
BL = new XON character (00h = current)
DX = port number (00h-03h) (04h-43h for XAPCM232.SYS)

**Return Registers:**

AH = status
        00h successful
        FFh error

**Conflicts:** Standard Function 05h (above), FOSSIL (below), MBBIOS (below), MultiDOS Plus (Chapter 45)

**See also:** Digiboard Function 04h

---

### INT 14h Function 08h

*Digiboard DigiCHANNEL PC/X**

### *ALTERNATE STATUS CHECK*

**Registers at call:**

AH = 08h
DX = port number (00h-03h)
(04h-43h for XAPCM232.SYS)

**Return Registers:**

AH = RS232 status bits (see Table 12-1)
ZF set if no characters queued
ZF clear if character available
        AL = next character

**Conflicts:** FOSSIL (below), MBBIOS (below), MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49)

**See also:** Digiboard Function 09h, Standard Function 03h (above), PC-MOS Function 08h (Chapter 49)

### INT 14h Function 09h
*Digiboard DigiCHANNEL PC/X**

***CLEAR BUFFERS***

**Registers at call:**
AH = 09h
DX = port number (00h-03h) (04h-43h for XAPCM232.SYS)

**Return Registers:**
AH = status (00h successful, FFh error)

**Conflicts:** FOSSIL (below), MBBIOS (below), MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49)

**See also:** Digiboard Functions 08h, 0Ah, and 10h

### INT 14h Function 0Ah
*Digiboard DigiCHANNEL PC/X**

***INPUT QUEUE CHECK***

**Registers at call:**
AH = 0Ah
DX = port number (00h-03h)
(04h-43h for XAPCM232.SYS)

**Return Registers:**
AX = number of characters available in buffer

**Details:** This function is also supported by the PC-MOS/386 v5.01 $serial.sys.

**Conflicts:** FOSSIL (below), MBBIOS (below)

**See also:** Digiboard Functions 09h and 0Dh

### INT 14h Function 0Dh
*Digiboard DigiCHANNEL PC/X**

***GET POINTER TO CH_KEY_RDY FLAG***

**Registers at call:**
AH = 0Dh
DX = port number (00h-03h)
(04h-43h for XAPCM232.SYS)

**Return Registers:**
ES:BX -> CH_KEY_RDY flag

**Conflicts:** FOSSIL (below)

**See also:** Digiboard Function 0Ah

**12-7 Values for CH_KEY_RDY flag:**

| | |
|---|---|
| 00h | receive buffer empty |
| FFh | characters available |

### INT 14h Function 0Eh
*Digiboard DigiCHANNEL PC/X**

***WRITE STRING***

**Registers at call:**
AH = 0EH
CX = number of characters to write
ES:BX -> string
DX = port number (00h-03h)
(04h-43h for XAPCM232.SYS)

**Return Registers:**
AX = number of characters actually written
ZF clear if successful
ZF set on error

**Conflicts:** FOSSIL (below)

**See also:** Digiboard Function 0Fh

### INT 14h Function 0Fh
*Digiboard DigiCHANNEL PC/X**

***READ STRING***

**Registers at call:**
AH = 0Fh
CX = number of characters to read
ES:BX -> buffer
DX = port number (00h-03h)
(04h-43h for XAPCM232.SYS)

**Return Registers:**
AX = number of characters read
ZF clear if successful
ZF set on error (line status or wrong number of characters)

**Conflicts:** FOSSIL (below)

**See also:** Digiboard Function 0Eh

---

### INT 14h Function 10h

*Digiboard DigiCHANNEL PC/X**

### *CLEAR RECEIVE BUFFER*

**Registers at call:**
AH = 10h
DX = port number (00h-03h)
(04h-43h for XAPCM232.SYS)

**Return Registers:**
AH = status
00h successful
FFh error

**Conflicts:** FOSSIL (below)
**See also:** Digiboard Functions 09h and 11h

---

### INT 14h Function 11h

*Digiboard DigiCHANNEL PC/X**

### *CLEAR TRANSMIT BUFFER*

**Registers at call:**
AH = 11h
DX = port number (00h-03h)
(04h-43h for XAPCM232.SYS)

**Return Registers:**
AH = status
00h successful
FFh error

**Conflicts:** FOSSIL (below), PC-MOS/386 (Chapter 49)
**See also:** Digiboard Functions 09h and 10h

---

### INT 14h Function 12h

*Digiboard DigiCHANNEL PC/X**

### *GET TRANSMIT BUFFER FREE SPACE*

**Registers at call:**
AH = 12h
DX = port number (00h-03h)
(04h-43h for XAPCM232.SYS)

**Return Registers:**
AX = number of bytes free

**Conflicts:** FOSSIL (below), PC-MOS/386 (Chapter 49)
**See also:** Digiboard Function 0Ah

---

### INT 14h Function 16h

*Digiboard DigiCHANNEL PC/X**

### *CCB COMMAND*

**Registers at call:**
AH = 16h
AL = CCB command number (see Table 12-8)
BL = byte 2
BH = byte 3
CL = byte 1 (for all channel functions except 4Eh and 4Fh)
DX = port number (00h-03h) (04h-43h for XAPCM232.SYS)

**Return Registers:**
AH = status
00h successful
FFh error

**Conflicts:** FOSSIL (below), PC-MOS/386 (Chapter 49)
**See also:** Digiboard Function 18h

**12-8 CCB command numbers:**

| | |
|---|---|
| 40h | Set Receive Mid Water Mark |
| 41h | Set Receive High Water Mark |
| 42h | Flush Receive Buffer |
| 43h | Flush Transmit Buffer |
| 44h | Transmit Pause |
| 45h | Transmit Resume |
| 46h | Set Interrupt to Host Mask |
| 47h | Set Baud, Data, Stop and Parity |
| 48h | Send Break |
| 49h | Set Modem Lines |
| 4Ah | Set Break Count |
| 4Bh | Set Handshake |

| | |
|---|---|
| 4Ch | Set Xon/Xoff Characters |
| 4Dh | Set Transmit Mid Water Mark |
| 4Eh | IRQ Polling Timer to Host |
| 4Fh | Buffer Set All |
| 50h | Port On |
| 51h | Port Off |
| 52h | Receive Pause |
| 53h | Special Character Interrupt |
| 54h | RS-422 Enable |

---

### INT 14h Function 18h

Digiboard DigiCHANNEL PC/X*

### *SEND BIOS COMMAND*

**Registers at call:**
AH = 18h
ES:BX -> 16-byte command string
DX = port number (00h-03h)
(04h-43h for XAPCM232.SYS)

**Return Registers:**
AH = status
(00h successful, 80h timeout)
AL = mailbox status
(00h no errors, 8Xh BIOS error)
ES:BX buffer filled in with mailbox string
ZF clear if no errors
ZF set if either status byte contains an error code

**Conflicts:** FOSSIL (below), PC-MOS/386 (Chapter 49)
**See also:** Digiboard Function 16h

---

### INT 14h Function 19h

Digiboard DigiCHANNEL PC/X*

### *SPECIAL CHARACTER INTERRUPT*

**Registers at call:**
AH = 19h
BL = flag
- 00h disable special character interrupt
- FFh enable interrupt

DX = port number (00h-03h) (04h-43h for XAPCM232.SYS)

**Return Registers:**
AH = status
- 00h successful
- FFh failed

**Conflicts:** FOSSIL (below), PC-MOS/386 (Chapter 49)
**See also:** Digiboard Function 1Ah

---

### INT 14h Function 1Ah

*Digiboard DigiCHANNEL PC/X*

### *SPECIAL CHARACTER FLAG/COUNTER*

**Registers at call:**
AH = 1Ah
BX = what to return
- 00h pointer to special character flag byte
- 01h pointer to special character counter word

DX = port number (00h-03h) (04h-43h for XAPCM232.SYS)

**Return Registers:**
ES:BX -> special character flag or counter

**Details:** The flag is FFh if one or more special characters are in the receive buffer; it is 00h and the counter is invalid if no special characters are in the receive buffer. The counter (if valid) contains the number of characters in the receive buffer up to and including the last-received special character.
**Conflicts:** FOSSIL (below)

## FOSSIL

FOSSIL is the Fido/Opus/Seadog Standard Interface Level, a standardized set of services originally designed to allow bulletin-board software to run on diverse hardware, including system whose serial ports were not IBM PC-compatible, without hard-coding the support into the BBS software.

## INT 11h Function BCh, Subfunction 1954h

BNU FOSSIL

### *INSTALLATION CHECK*

**Registers at call:**
AH = BCh
DX = 1954h

**Return Registers:**
AX = 1954h
ES:DX -> entry point of driver (instead of INT 14h)

**Conflicts:** Alignment Check (Chapter 3), Standard INT 11h (Chapter 4), SDLP (Chapter 11), Blank-It (Chapter 14), Back and Forth (Chapter 49), Borland C++ IDE (Chapter 54), RainboxFAX (Chapter 64)
**See also:** FOSSIL INT 14h Function 04h

---

## INT 14h Function 00h

FOSSIL

### *INITIALIZE*

**Registers at call:**
AH = 00h
AL = initializing parameters
    bits 7-5 data rate (19200,38400,300, 600,1200,2400,4800,9600 bps)
    bits 4-3 parity (00 or 10 = none, 01 = odd, 11 = even)
    bit 2 stop bits (set = 2, clear = 1)
    bits 1-0 data bits (00 = 5, 01 = 6, 10 = 7, 11 = 8)
DX = port number (0-3 or FFh if only performing non-I/O setup)

**Return Registers:**
AH = RS-232 status code bits
    bit 0: RDA - input data is available in buffer
    bit 1: OVRN - data has been lost
    bit 5: THRE - room is available in output buffer
    bit 6: TSRE - output buffer empty
AL = modem status bits
    bit 3: always 1
    bit 7: DCD - carrier detect

**Conflicts:** Standard Function 00h (above), MBBIOS (below)
**See also:** FOSSIL Function 05h, COMM-DRV Function 81h (below), COURIERS Function 82h (below)

---

## INT 14h Function 02h

FOSSIL

### *RECEIVE CHARACTER WITH WAIT*

**Registers at call:**
AH = 02h
DX = port number (0-3)

**Return Registers:**
AL = character received
AH = 00h

**Conflicts:** Standard Function 02h (above)
**See also:** Function 01h, Standard Function 02h (above)

---

## INT 14h Function 04h

FOSSIL

### *INITIALIZE DRIVER*

**Registers at call:**
AH = 04h
DX = port number
optionally, if BX=4F50h
    ES:CX -> byte to be set upon ^C

**Return Registers:**
AX = 1954h (if successful)
BL = maximum function number supported (excluding 7Eh and above)
BH = revision of FOSSIL specification supported
DTR is raised

**Details:** The word at offset 6 in the interrupt handler contains 1954h, and the following byte contains the maximum function number supported; this can serve as an installation check.
**Conflicts:** Standard Function 04h (above), Digiboard (above), MBBIOS (below), TSRCOMM (below), MultiDOS Plus (Chapter 45)
**See also:** Functions 05h and 1Ch, INT 11h Function BCh

### INT 14h Function 05h
FOSSIL

*DEINITIALIZE DRIVER*

**Registers at call:**
AH = 05h
DX = port number

**Return Registers:**
none
DTR is not affected

**Conflicts:** Standard Function 04h (above), Digiboard (above), MBBIOS (below), MultiDOS Plus (Chapter 45)

**See also:** Functions 00h, 04h,and 1Dh; COURIERS Function 8Dh (below)

### INT 14h Function 06h
FOSSIL

*RAISE/LOWER DTR*

**Registers at call:**
AH = 06h
DX = port
AL = DTR state to be set
  00h lower, 01h raise

**Return Registers:**
none

**Conflicts:** MBBIOS (below), MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49), TelAPI (NI)

**See also:** Function 1Ah, MBBIOS Function 05h (below)

### INT 14h Function 07h
FOSSIL

*RETURN TIMER TICK PARAMETERS*

**Registers at call:**
AH = 07h

**Return Registers:**
AL = timer tick interrupt number
AH = ticks per second on interrupt number in AL
DX = approximate number of milliseconds per tick

**Conflicts:** MBBIOS (below), MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49), TelAPI (NI)

**See also:** Function 16h

### INT 14h Function 08h
FOSSIL

*FLUSH OUTPUT BUFFER WAITING TILL ALL OUTPUT IS DONE*

**Registers at call:**
AH = 08h
DX = port number

**Return Registers:**
none

**Conflicts:** Digiboard (above), MBBIOS (below), MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49)

**See also:** Function 09h

### INT 14h Function 09h
FOSSIL

*PURGE OUTPUT BUFFER THROWING AWAY ALL PENDING OUTPUT*

**Registers at call:**
AH = 09h
DX = port number

**Return Registers:**
none

**Conflicts:** Digiboard (above), MBBIOS (below), MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49)

**See also:** Functions 08h and 0Ah, COURIERS Function 88h (below)

### INT 14h Function 0Ah
FOSSIL

*PURGE INPUT BUFFER THROWING AWAY ALL PENDING INPUT*

**Registers at call:**
AH = 0Ah
DX = port number

**Return Registers:**
none

**Conflicts:** Digiboard (above), MBBIOS (below)
**See also:** Function 09h, COURIERS Function 85h (below)

---

## INT 14h Function 0Bh
### *TRANSMIT NO WAIT*

*FOSSIL*

**Registers at call:**
AH = 0Bh
AL = character
DX = port number

**Return Registers:**
AX = status
0000h character not accepted
0001h character accepted

**Conflicts:** MBBIOS (below)
**See also:** Function 01h

---

## INT 14h Function 0Ch
### *NON-DESTRUCTIVE READ AHEAD*

*FOSSIL*

**Registers at call:**
AH = 0Ch
DX = port number

**Return Registers:**
AX = FFFFh character not available
AX = 00xxh character xx available

**See also:** Function 20h, MBBIOS Function 08h (below)

---

## INT 14h Function 0Dh
### *KEYBOARD READ WITHOUT WAIT*

*FOSSIL*

**Registers at call:**
AH = 0Dh

**Return Registers:**
AX = result
FFFFh character not available
xxyyh standard IBM-style scan code

**Conflicts:** Digiboard (above)
**See also:** Function 0Eh

---

## INT 14h Function 0Eh
### *KEYBOARD READ WITH WAIT*

*FOSSIL*

**Registers at call:**
AH = 0Eh

**Return Registers:**
AX = xxyyh standard IBM-style scan code

**Conflicts:** Digiboard (above)
**See also:** Function 0Dh

---

## INT 14h Function 0Fh
### *ENABLE/DISABLE FLOW CONTROL*

*FOSSIL*

**Registers at call:**
AH = 0Fh
AL = bit mask describing requested flow control (see Table 12-9)
DX = port number

**Return Registers:**
none

**Conflicts:** Digiboard (above)
**See also:** Function 10h, MBBIOS Function 09h (below)

**12-9 Bitfields for requested flow control:**

| | |
|---|---|
| bit 0 | XON/XOFF on transmit (watch for XOFF while sending) |
| bit 1 | CTS/RTS (CTS on transmit/RTS on receive) |
| bit 2 | reserved |
| bit 3 | XON/XOFF on receive (send XOFF when buffer near full) |
| bit 4-7 | all 1 |

---

### INT 14h Function 10h

*FOSSIL*

### *EXTENDED ^C/^K CHECKING AND TRANSMIT ON/OFF*

**Registers at call:**
AH = 10h
AL = bit mask
    bit 0: enable/disable ^C/^K checking
    bit 1: enable/disable the transmitter
DX = port number

**Return Registers:**
none

**Conflicts:** Digiboard (above)
**See also:** Function 0Fh

---

### INT 14h Function 11h

*FOSSIL*

### *SET CURRENT CURSOR LOCATION*

**Registers at call:**
AH = 11h
DH = row
DL = column

**Return Registers:**
none

**Details:** This is the same as INT 10h Function 02h.
**Conflicts:** Digiboard (above), PC-MOS/386 (Chapter 49)
**See also:** Function 12h

---

### INT 14h Function 12h

*FOSSIL*

### *READ CURRENT CURSOR LOCATION*

**Registers at call:**
AH = 12h

**Return Registers:**
DH = row
DL = column

**Details:** This is the same as INT 10h Function 03h.
**Conflicts:** Digiboard (above), PC-MOS/386 (Chapter 49)
**See also:** Function 11h

---

### INT 14h Function 13h

*FOSSIL*

### *SINGLE CHARACTER ANSI WRITE TO SCREEN*

**Registers at call:**
AH = 13h
AL = character

**Return Registers:**
none

**Details:** Should not be called if it is unsafe to call DOS.
**Conflicts:** PC-MOS/386 (Chapter 49)
**See also:** Function 15h

---

### INT 14h Function 14h

*FOSSIL*

### *ENABLE OR DISABLE WATCHDOG PROCESSING*

**Registers at call:**
AH = 14h
AL = 01h enable watchdog
    00h disable watchdog
DX = port number

**Return Registers:**
none

**Conflicts:** PC-MOS/386 (Chapter 49)

---

### INT 14h Function 15h

*FOSSIL*

### *WRITE CHARACTER TO SCREEN USING BIOS SUPPORT ROUTINES*

**Registers at call:**
AH = 15h
AL = character

**Return Registers:**
none

**Conflicts:** PC-MOS/386 (Chapter 49)
**See also:** Function 13h

---

### INT 14h Function 16h
*FOSSIL*

### *INSERT/DELETE FUNCTION FROM TIMER TICK CHAIN*

**Registers at call:**
AH = 16h
AL = function
    00h = delete
    01h = add
ES:DX -> routine to call

**Return Registers:**
AX = status
    0000h successful
    0001h unsuccessful

**Conflicts:** Digiboard (above), PC-MOS/386 (Chapter 49)
**See also:** FOSSIL Function 07h

---

### INT 14h Function 17h
*FOSSIL*

### *REBOOT SYSTEM*

**Registers at call:**
AH = 17h
AL = method
    00h = cold boot
    01h = warm boot

**Return Registers:**
none

**Conflicts:** PC-MOS/386 (Chapter 49)
**See also:** INT 19h

---

### INT 14h Function 18h
*FOSSIL*

### *READ BLOCK*

**Registers at call:**
AH = 18h
CX = maximum number of characters to transfer
DX = port number
ES:DI -> user buffer

**Return Registers:**
AX = number of characters transferred

**Conflicts:** Digiboard (above), PC-MOS/386 (Chapter 49)
**See also:** Function 19h, COURIERS Function 83h (below), EBIOS Function FF02h (below), NASI INT 6Bh Function 0100h (NI)

---

### INT 14h Function 19h
*FOSSIL*

### *WRITE BLOCK*

**Registers at call:**
AH = 19h
CX = maximum number of characters to transfer
DX = port number
ES:DI -> user buffer

**Return Registers:**
AX = number of characters transferred

**Conflicts:** Digiboard (above), PC-MOS/386 (Chapter 49)
**See also:** Function 18h, COURIERS Function 86h (below), NASI INT 6Bh Function 0000h (NI)

---

### INT 14h Function 1Ah
*FOSSIL*

### *BREAK BEGIN OR END*

**Registers at call:**
AH = 1Ah
AL = 00h stop sending 'break'
    01h start sending 'break'
DX = port number

**Return Registers:**
none

**Conflicts:** Digiboard (above)
**See also:** Function 06h, COURIERS Function 8Ah (below), EBIOS Function FAh (below)

---

## INT 14h Function 1Bh

*FOSSIL*

### *RETURN INFORMATION ABOUT THE DRIVER*

**Registers at call:**
AH = 1Bh
DX = port number
CX = size of user buffer
ES:DI -> user buffer for driver info
(see Table 12-10)

**Return Registers:**
AX = number of characters transferred
CX = 3058h ("0X") (X00 FOSSIL only)
DX = 2030h (" 0") (X00 FOSSIL only)

**12-10 Format of driver info:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | size of structure in bytes |
| 02h | BYTE | FOSSIL spec driver conforms to |
| 03h | BYTE | revision level of this specific driver |
| 04h | DWORD | pointer to ASCIZ identification string |
| 08h | WORD | size of the input buffer |
| 0Ah | WORD | number of bytes left in buffer |
| 0Ch | WORD | size of the output buffer |
| 0Eh | WORD | number of bytes left in buffer |
| 10h | BYTE | width of screen |
| 11h | BYTE | length of screen |
| 12h | BYTE | actual baud rate, computer to modem |

---

## INT 14h Function 1Ch

*X00 FOSSIL*

### *ACTIVATE PORT*

**Registers at call:**
AH = 1Ch
DX = port number

**Return Registers:**
AX = 1954h if successful
BL = maximum function number supported
(not including 7Eh and above)
BH = revision of FOSSIL specification supported

**Details:** This is a duplicate of Function 04h, so that Function 04h may be made compatible with the PS/2 BIOS in a future release.
**See also:** Functions 04h and 1Dh

---

## INT 14h Function 1Dh

*X00 FOSSIL*

### *DEACTIVATE PORT*

**Registers at call:**
AH = 1Dh
DX = port number

**Return Registers:**
none

**Details:** This is a duplicate of Function 05h, so that Function 05h may be made compatible with the PS/2 BIOS in a future release. Ignored if the port was never activated with Functions 04h or 1Ch.
**See also:** Functions 05h and 1Ch

---

## INT 14h Function 1Eh

*X00 FOSSIL*

### *EXTENDED LINE CONTROL INITIALIZATION*

**Registers at call:**
AH = 1Eh
AL = break status
00h if break

01h if no break
BH = parity (see Table 12-11)
BL = number of stop bits
00h one stop bit
01h two stop bits (1.5 if 5 bit word length)
CH = word length (see Table 12-12)
CL = bps rate (see Table 12-13)
DX = port number

**Return Registers:**
AX = port status code
(see Tables 12-1 and 12-2)

**Details:** This function is intended to exactly emulate the PS/2 BIOS Function 04h call.

If the port was locked at X00 load time, the appropriate parameters are ignored.

**See also:** Function 00h, Standard Function 04h (above)

**12-11 Values for parity:**

| | |
|---|---|
| 00h | no parity |
| 01h | odd parity |
| 02h | even parity |
| 03h | stick parity odd |
| 04h | stick parity even |

**12-12 Values for word length:**

| | |
|---|---|
| 00h | 5 bits |
| 01h | 6 bits |
| 02h | 7 bits |
| 03h | 8 bits |

**12-13 Values for bps rate:**

| | |
|---|---|
| 00h | 110 |
| 01h | 150 |
| 02h | 300 |
| 03h | 600 |
| 04h | 1200 |
| 05h | 2400 |
| 06h | 4800 |
| 07h | 9600 |
| 08h | 19200 |

## INT 14h Function 1Fh

*X00 FOSSIL*

### *EXTENDED SERIAL PORT STATUS/CONTROL*

**Registers at call:**
AH = 1Fh
DX = port number
AL = function
00h read modem control register

**Return Registers:**
BL = modem control register
(see Table 12-14)
AH = status

01h write modem control register
BL = modem control register (see Table 12-14)

AX = status

**Details:** This function is intended to exactly emulate the PS/2 BIOS Function 05h call.

X00 forces BL bit 3 set (interrupts cannot be disabled).

**See also:** Function 00h, Standard Function 05h (above)

**12-14 Bitfields for modem control register:**

| | |
|---|---|
| bit 0 | data terminal ready |
| bit 1 | request to send |

| | |
|---|---|
| bit 2 | OUT1 |
| bit 3 | OUT2 (interrupts) enabled |
| bit 4 | LOOP |
| bits 5-7 | reserved |

---

### INT 14h Function 20h

*X00 FOSSIL*

### *DESTRUCTIVE READ WITH NO WAIT*

**Registers at call:**
AH = 20h
DX = port number

**Return Registers:**
AH = 00h if character was available
    AL = next character (removed from receive buffer)
AX = FFFFh if no character available

**Conflicts:** MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49), Alloy MW386 (NI)
**See also:** Functions 0Ch and 21h

---

### INT 14h Function 21h

*X00 FOSSIL*

### *STUFF RECEIVE BUFFER*

**Registers at call:**
AH = 21h
AL = character
DX = port number

**Return Registers:**
none

**Details:** The given character is inserted at the end of the receive buffer as if it had just arrived from the serial port; all normal receive processing (XON/XOFF, ^C/^K) is performed on the character.

This function is fully re-entrant.

**Conflicts:** MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49), Alloy MW386 (NI)
**See also:** Function 20h

---

### INT 14h Function 7Eh

*FOSSIL*

### *INSTALL AN EXTERNAL APPLICATION FUNCTION*

**Registers at call:**
AH = 7Eh
AL = code assigned to external application (80h-BFh)
    80h reserved for communications FOSSIL
    81h video FOSSIL
    82h reserved for keyboard FOSSIL
    83h reserved for system FOSSIL
ES:DX -> entry point

**Return Registers:**
AX = 1954h
BL = code assigned to application (same as input AL)
DH = status
    00h failed
    01h successful

**See also:** Function 7Fh,FOSSIL Function 80h,Function 8100h,FOSSIL Function 82h,FOSSIL Function 83h

---

### INT 14h Function 7Fh

*FOSSIL*

### *REMOVE AN EXTERNAL APPLICATION FUNCTION*

**Registers at call:**
AH = 7Fh
AL = code assigned to external application
ES:DX -> entry point

**Return Registers:**
AX = 1954h
BL = code assigned to application (same as input AL)
DH = status
    00h failed
    01h successful

**See also:** Function 7Eh

### INT 14h Function 80h

COMMUNICATIONS FOSSIL

**Registers at call:**
AH = 80h

**Return Registers:**
vary

**Conflicts:** COURIERS.COM (below), ARTICOM (NI)
**See also:** Function 7Eh

### INT 14h Function 81h
### *VIDEO FOSSIL API—see Chapter 8*

VIDEO FOSSIL

### INT 14h Function 83h
### *API*

SYSTEM FOSSIL

**Registers at call:**
AH = 83h

**Return Registers:**
vary

**Conflicts:** COURIERS.COM (below)
**See also:** Function 7Eh

### INT 14h Function E000h
### *GET MNP STATUS BLOCK*

MX5 Extended FOSSIL

**Registers at call:**
AX = E000h
DX = port number (0-3)

**Return Registers:**
ES:BX -> status block (see Table 12-15)

**Program:** MX5 is a FOSSIL driver by MagicSoft which emulates MNP Level 5, and ships with the MTEZ terminal program as MTEMNP.DRV (a TSR despite the .DRV extension)
**Conflicts:** TelAPI (NI)
**See also:** Function E006h

**12-15 Format of status block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | flag: active (00h no, 01h yes) |
| 01h | BYTE | MNP level (2,4,5) |
| 02h | BYTE | series ID from remote MNP |
| 03h | DWORD | total packets transmitted |
| 07h | DWORD | duplicate packets transmitted |
| 0Bh | DWORD | maximum speed |
| 0Fh | DWORD | total packets received |
| 13h | DWORD | duplicate packets received |
| 17h | DWORD | maximum speed |

### INT 14h Function E001h
### *GET/SET MNP LEVEL*

MX5 Extended FOSSIL

**Registers at call:**
AX = E001h
BH = function
    00h get MNP level
    01h set MNP level
        BL = new level (00h none, 02h/04h/05h MNP level N)
DX = port number (0-3)

**Return Registers:**
BL = MNP level

**Conflicts:** TelAPI (NI)
**See also:** Functions E002h, E003h, E004h, and E006h

## INT 14h Function E002h

*MX5 Extended FOSSIL*

### *GET/SET MNP ANSWER/ORIGINATE MODE*

**Registers at call:**
AX = E002h
BH = function
00h get answer/originate mode
01h set mode
BL = new mode (00h originate [default], 01h answer)
DX = port number (0-3)

**Return Registers:**
BL = answer/originate mode

**Conflicts:** TelAPI (NI)

**See also:** Functions E001h, E003h, and E006h

## INT 14h Function E003h

*MX5 Extended FOSSIL*

### *GET/SET MNP WAIT TICKS*

**Registers at call:**
AX = E003h
BH = function
00h get wait ticks
01h set wait ticks
BL = MNP wait ticks (default 0Eh)
DX = port number (0-3)

**Return Registers:**
BL = wait ticks

**Conflicts:** TelAPI (NI)

**See also:** Functions E001h, E002h, and E006h

## INT 14h Function E004h

*MX5 Extended FOSSIL*

### *GET/SET MNP CONNECT SOUND LEVEL*

**Purpose:** Specify whether MX5 should generate beeps after an MNP connection (three high beeps if successful, high then low on connection failure).

**Registers at call:**
AX = E004h
BH = function
00h get sound level
01h set sound level
BL = new sound level (00h off, 01h on [default])
DX = port number

**Return Registers:**
BL = sound state

**Conflicts:** TelAPI (NI)

**See also:** Functions E002h and E006h

## INT 14h Function E005h

*MX5 Extended FOSSIL*

### *UNINSTALL*

**Registers at call:**
AX = E005h

**Return Registers:**
BX = segment of MX5's memory block or 0000h on failure

**Details:** The caller must free the returned memory block to complete the uninstall.

**Conflicts:** TelAPI (NI)

**See also:** Function E006h

## INT 14h Function E006h

*MX5 Extended FOSSIL*

### *INSTALLATION CHECK*

**Registers at call:**
AX = E006h
BX = 0000h

**Return Registers:**
BX = 4D58h ('MX') if installed

AH = major version
AL = minor version

**Conflicts:** TelAPI (NI)
**See also:** Functions E000h, E001h, E005h, and E007h

---

### INT 14h Function E007h

*MX5 Extended FOSSIL*

### *WAIT SPECIFIED NUMBER OF TICKS*

**Registers at call:**
AX = E007h
CX = number of ticks to wait

**Return Registers:**
none

**Conflicts:** TelAPI (NI)
**See also:** Function E006h

## MBBIOS

MBBIOS was written by H. Roy Engehausen.

---

### INT 14h Function 00h

*MBBIOS*

### *INITIALIZE PORT*

**Registers at call:**
AH = 00h
AL = port parameters
    bits 7-5 data rate
        (normally 110,150,300,600,1200,2400,4800,9600 bps;
        9600,14400,19200,28800,38400,57600,115200,330400 bps
        if the high-speed option is set)
    bits 4-3 parity (00 or 10 = none, 01 = odd, 11 = even)
    bit 2 stop bits (set = 2, clear = 1)
    bits 1-0 data bits (00 = 5, 01 = 6, 10 = 7, 11 = 8)
DX = port number

**Return Registers:**
AH = line status (see Table 12-1)
AL = modem status (see Table 12-2)

**Conflicts:** Standard Function 00h (above), FOSSIL Function 00h (above)
**See also:** Functions 04h and 05h

---

### INT 14h Function 04h

*MBBIOS*

### *INQUIRY*

**Registers at call:**
AH = 04h

**Return Registers:**
AH = AAh
AL = 55h

**Conflicts:** Standard Function 04h (above), Digiboard (above), FOSSIL (above), TSRCOMM (below), MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49)
**See also:** Functions 00h and 09h

---

### INT 14h Function 05h

*MBBIOS*

### *DROP DTR AND RTS*

**Registers at call:**
AH = 05h
DX = port number

**Return Registers:**
none

**Conflicts:** Standard Function 04h (above), Digiboard (above), FOSSIL (above), MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49)
**See also:** Functions 00h and 06h, FOSSIL Function 06h (above)

---

### INT 14h Function 06h

*MBBIOS*

### *RAISE DTR AND RTS*

**Registers at call:**
AH = 06h
DX = port number

**Return Registers:**
none

**Conflicts:** FOSSIL (above), MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49), TelAPI (NI)
**See also:** Functions 05h and 07h

---

### INT 14h Function 07h
*MBBIOS*
### *SEND BREAK*

**Registers at call:**
AH = 07h
DX = port number

**Return Registers:**
none

**Conflicts:** FOSSIL (above), MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49), TelAPI (NI)
**See also:** Function 06h

---

### INT 14h Function 08h
*MBBIOS*
### *NON-DESTRUCTIVE READ*

**Registers at call:**
AH = 08h
DX = port number

**Return Registers:**
AL = character
AH = status (see Table 12-1)

**Conflicts:** FOSSIL (above), MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49)
**See also:** Function 0Bh, FOSSIL Function 0Ch (above)

---

### INT 14h Function 09h
*MBBIOS*
### *GET/SET OPTIONS*

**Registers at call:**
AH = 09h
AL = option byte
    00000001b - Transmit Buffering Enabled
    00000101b - Hardware Handshaking Enabled
    00100000b - High Speed Option Enabled
DX = *port number*

**Return Registers:**
AL = old option byte

Conflicts: Digiboard, FOSSIL (above), MultiDOS Plus (Chapter 45), PC-MOS/386 (Chapter 49)
**See also:** Function 04h, FOSSIL Function 10h (above)

---

### INT 14h Function 0Ah
*MBBIOS*
### *WRITE BUFFER*

**Registers at call:**
AH = 0Ah
CX = count
ES:DI -> buffer

**Return Registers:**
AX = status (see Tables 12-1 and 12-2)
DI = is updated
CX = unsent character count

**Conflicts:** Digiboard (above), FOSSIL (above)
**See also:** Function 0Bh, FOSSIL Function 19h (above)

---

### INT 14h Function 0Bh
*MBBIOS*
### *READ BUFFER*

**Registers at call:**
AH = 0Bh
CX = size of buffer
ES:DI -> buffer

**Return Registers:**
AH = LSR
AL = MSR
CX = count of characters
DI updated

**Conflicts:** FOSSIL (above)
**See also:** Functions 08h and 0Ah, FOSSIL Function 18h (above)

## AVATAR Serial Dispatcher

The AVATAR Serial Dispatcher is an enhancement to AVATAR.SYS (see Chapter 8) which allows multiple programs to share a single serial port, even concurrently.

---

### INT 2Fh Function 1A42h

*AVATAR Serial Dispatcher*

***INSTALL IRQ3 HANDLER***

**Registers at call:**
AX = 1A42h
BX = 4156h ('AV')
ES:DI -> FAR handler for serial port using IRQ3
DS = data segment needed by handler

**Return Registers:**
AX = status
1A42h if ASD not installed
0000h if no more room
else handle to use when uninstalling

**Details:** The handler need not save/restore registers or signal EOI to the interrupt controller. The handler should return AX=0000h if the interrupt was meant for it, and either leave AX unchanged or return a non-zero value otherwise. The most recently installed handler will be called first, continuing to earlier handlers until one returns AX=0000h.

**See also:** Functions 1A43h and 1A62h

---

### INT 2Fh Function 1A43h

*AVATAR Serial Dispatcher*

***INSTALL IRQ4 HANDLER***

**Registers at call:**
AX = 1A43h
BX = 4156h ('AV')
ES:DI -> FAR handler for serial port using IRQ4
DS = data segment needed by handler

**Return Registers:**
AX = result
1A43h if ASD not installed
0000h if no more room
else handle to use when uninstalling

**Details:** (see Function 1A42h).

**See also:** Functions 1A42h and 1A63h

---

### INT 2Fh Function 1A62h

*AVATAR Serial Dispatcher*

***UNINSTALL IRQ3 HANDLER***

**Registers at call:**
AX = 1A62h
BX = 4156h ('AV')
CX = handle for IRQ routine returned by Function 1A42h

**Return Registers:**
none

**See also:** Functions 1A42h and 1A63h

---

### INT 2Fh Function 1A63h

*AVATAR Serial Dispatcher*

***UNINSTALL IRQ4 HANDLER***

**Registers at call:**
AX = 1A63h
BX = 4156h ('AV')
CX = handle for IRQ routine returned by Function 1A43h

**Return Registers:**
none

**See also:** Functions 1A43h and 1A62h

## COURIERS.COM

COURIERS is a TSR utility by *PC* Magazine.

---

### INT 14h Function 80h

*COURIERS.COM*

***INSTALLATION CHECK***

**Registers at call:**
AH = 80h

**Return Registers:**
AH = E8h if loaded

**Conflicts:** Communications FOSSIL (above), ARTICOM (NI)

**INT 14h Function 81h** *COURIERS.COM*

***CHECK IF PORT BUSY***

**Registers at call:**
AH = 81h
AL = port number (1-4)

**Return Registers:**
AH = status
  00h port available
  01h port exists but already in use
  02h port nonexistent

**Conflicts** COMM-DRV (below), Video FOSSIL (Chapter 8)
**See also:** Functions 83h and 8Dh

---

**INT 14h Function 82h** *COURIERS.COM*

***CONFIGURE PORT***

**Registers at call:**
AH = 82h
AL = port number (1-4)
BX = speed (bps)
CX = bit flags
  bit 0: enable input flow control
  bit 1: enable output flow control
  bit 2: use X.PC protocol (not yet implemented)

**Return Registers:**
none

**Conflicts:** Keyboard FOSSIL (Chapter 13)
**See also:** Function 8Ch, standard Function 00h (above)

---

**INT 14h Function 83h** *COURIERS.COM*

***START INPUT***

**Registers at call:**
AH = 83h
ES:BX -> circular input buffer
CX = length of buffer

**Return Registers:**
none

**Details:** The input buffer should be at least 128 bytes if input flow control is enabled.
**Conflicts:** System FOSSIL (above)
**See also:** Functions 87h and 8Dh, FOSSIL Function 18h (above), BAPI Function A5h (NI)

---

**INT 14h Function 84h** *COURIERS.COM*

***READ CHARACTER***

**Registers at call:**
AH = 84h

**Return Registers:**
ZF set if no characters available
ZF clear
  AL = character
  AH = modem status bits
    bit 7: set on input buffer overflow

**See also:** Functions 86h and 89h, standard Function 02h (above)

---

**INT 14h Function 85h** *COURIERS.COM*

***FLUSH PENDING INPUT***

**Registers at call:**
AH = 85h

**Return Registers:**
none

**See also:** Function 0Ah,Function 88h

---

**INT 14h Function 86h** *COURIERS.COM*

***START OUTPUT***

**Registers at call:**
AH = 86h

**Return Registers:**
ES:BX -> output buffer
none
CX = length of output buffer

**See also:** Function 83h, FOSSIL Function 19h (above), BAPI Function A4h (NI)

---

### INT 14h Function 87h
*COURIERS.COM*

### *OUTPUT STATUS*

**Registers at call:**
AH = 87h

**Return Registers:**
AX = number of unsent characters

---

### INT 14h Function 88h
*COURIERS.COM*

### *ABORT OUTPUT*

**Registers at call:**
AH = 88h

**Return Registers:**
none

**See also:** Function 85h, FOSSIL Function 09h (above)

---

### INT 14h Function 89h
*COURIERS.COM*

### *SEND SINGLE CHARACTER*

**Registers at call:**
AH = 89h
CL = character to send

**Return Registers:**
none

**See also:** Function 84h, standard Function 01h (above)

---

### INT 14h Function 8Ah
*COURIERS.COM*

### *SEND BREAK*

**Registers at call:**
AH = 8Ah

**Return Registers:**
none

**See also:** Function 89h, EBIOS Function FAh (below)

---

### INT 14h Function 8Ch
*COURIERS.COM*

### *SET SPEED*

**Registers at call:**
AH = 8Ch
BX = speed in bps

**Return Registers:**
none

**See also:** Function 82h, standard Function 00h (above)

---

### INT 14h Function 8Dh
*COURIERS.COM*

### *DECONFIGURE PORT*

**Registers at call:**
AH = 8Dh

**Return Registers:**
none

**See also:** Function 82h

## IBM/Yale EBIOS

---

### INT 14h Function F4FFh
*IBM/Yale EBIOS SERIAL I/O*

### *INSTALLATION CHECK*

**Registers at call:**
AX = F4FFh
DX = port (00h-03h)

**Return Registers:**
CF clear if present
  AX = 0000h
CF set if not present
  AX nonzero

**See also:** Function F9h

### INT 14h Function F9h
*IBM/Yale EBIOS SERIAL I/O*

***REGAIN CONTROL***

**Registers at call:**
AH = F9h
DX = port (00h-03h)

**Return Registers:**
none

### INT 14h Function FAh
*IBM/Yale EBIOS SERIAL I/O*

***SEND BREAK***

**Registers at call:**
AH = FAh
DX = port (00h-03h)

**Return Registers:**
none

**See also:** FOSSIL Function 1Ah (above), COURIERS Function 8Ah (above)

### INT 14h Function FBh
*IBM/Yale EBIOS SERIAL I/O*

***SET OUTGOING MODEM SIGNALS***

**Registers at call:**
AH = FBh
AL = modem control register (see Table 12-6) AX = port (00h-03h)

**Return Registers:**
none

**See also:** Standard Function 05h (above)

### INT 14h Function FCh
*IBM/Yale EBIOS SERIAL I/O*

***READ CHARACTER, NO WAIT***

**Registers at call:**
AH = FCh
DX = port (00h-03h)

**Return Registers:**
AH = RS232 status bits (see Table 12-1)
AL = character

**See also:** Function FF02h, Standard Function 02h (above), FOSSIL Function 0Ch (above)

### INT 14h Function FD02h
*IBM/Yale EBIOS SERIAL I/O*

***READ STATUS***

**Registers at call:**
AX = FD02h

**Return Registers:**
CX = number of characters available

### INT 14h Function FF02h
*IBM/Yale EBIOS SERIAL I/O*

***BUFFERED READ***

**Registers at call:**
AX = FF02h
CX = length
DX = port (00h-03h)
ES:BX -> buffer

**Return Registers:**
CX = number of characters read

**Conflicts:** TelAPI (NI)

**See also:** Function FCh, FOSSIL Function 18h (above), COURIERS Function 83h (above), BAPI Function A5h (NI)

## TSRCOMM

TSRCOMM is a program from Microsoft Systems Journal which provides interrupt-driven I/O on INT 14h.

### INT 14h Function 0400h
*TSRCOMM*

***INSTALLATION CHECK***

**Registers at call:**
AX = 0400h

**Return Registers:**
AX = 0FF0h

**Conflicts:** Standard Function 04h (above), Digiboard (above), FOSSIL (above), MBBIOS (above), MultiDOS Plus (Chapter 45)
**See also:** Functions 0401h and 0408h

---

### INT 14h Function 0401h
*TSRCOMM*

***INITIALIZE MODE***

**Registers at call:**
AX = 0401h
CX = mode

**Return Registers:**
none

**Conflicts:** Standard Function 04h (above), Digiboard (above), FOSSIL (above), MBBIOS (above), MultiDOS Plus (Chapter 45)
**See also:** Functions 0400h and 0402h

---

### INT 14h Function 0402h
*Microsoft Systems Journal TSRCOMM INT14*

***EXTENDED INITIALIZE***

**Registers at call:**
AX = 0402h
CL = parameters

**Return Registers:**
none

**Conflicts:** Standard Function 04h (above), Digiboard (above), FOSSIL (above), MBBIOS (above), MultiDOS Plus (Chapter 45)
**See also:** Functions 0400h and 0401h

---

### INT 14h Function 0403h
*Microsoft Systems Journal TSRCOMM INT14*

***SET TIMEOUT***

**Registers at call:**
AX = 0403h
CX = timeout

**Return Registers:**
none

**Conflicts:** Standard Function 04h (above), Digiboard (above), FOSSIL (above), MBBIOS (above), MultiDOS Plus (Chapter 45)
**See also:** Function 0400h

---

### INT 14h Function 0404h
*Microsoft Systems Journal TSRCOMM INT14*

***CLEAR THE RECEIVE BUFFER***

**Registers at call:**
AX = 0404h

**Return Registers:**
none

**Conflicts:** Standard Function 04h (above), Digiboard (above), FOSSIL (above), MBBIOS (above), MultiDOS Plus (Chapter 45)
**See also:** Functions 0400h, 0405h, and 0406h

---

### INT 14h Function 0405h
*Microsoft Systems Journal TSRCOMM INT14*

***GET RECEIVE BUFFER COUNT***

**Registers at call:**
AX = 0405h

**Return Registers:**
AX = number of characters in buffer

**Conflicts:** Standard Function 04h (above), Digiboard (above), FOSSIL (above), MBBIOS (above), MultiDOS Plus (Chapter 45)
**See also:** Functions 0400h, 0404h, and 0407h

---

### INT 14h Function 0406h
*Microsoft Systems Journal TSRCOMM INT14*

***CLEAR THE TRANSMIT BUFFER***

**Registers at call:**
AX = 0406h

**Return Registers:**
none

**Conflicts:** Standard Function 04h (above), Digiboard (above), FOSSIL (above), MBBIOS (above), MultiDOS Plus (Chapter 45)
**See also:** Functions 0400h, 0404h, and 0407h

### INT 14h Function 0407h
*Microsoft Systems Journal TSRCOMM INT14*

### *GET TRANSMIT BUFFER COUNT*

**Registers at call:**
AX = 0407h

**Return Registers:**
AX = number of characters in the buffer

**Conflicts:** Standard Function 04h (above), Digiboard (above), FOSSIL (above), MBBIOS (above), MultiDOS Plus (Chapter 45)

**See also:** Functions 0400h, 0405h, and 0406h

### INT 14h Function 0408h
*Microsoft Systems Journal TSRCOMM INT14*

### *UNINSTALL*

**Registers at call:**
AX = 0408h

**Return Registers:**
none

**Conflicts:** Standard Function 04h (above), Digiboard (above), FOSSIL (above), MBBIOS (above), MultiDOS Plus (Chapter 45)

**See also:** Function 0400h

## Miscellaneous

### INT 14h Function 81h
*Egberto Willies COMM-DRV*

### *EXTENDED INITIALIZATION*

**Registers at call:**
AH = 81h
BX:DI -> port control block

**Return Registers:**
*unknown*

**Conflicts:** COURIERS (above), Video FOSSIL (Chapter 8)

**See also:** Standard Function 00h (above)

**12-16 Format of port control block:**

| *Offset* | *Type* | *Description* |
|---|---|---|
| 00h | WORD | port IO address |
| 02h | WORD | port IRQ |
| 04h | WORD | baud rate |
| 06h | WORD | parity |
| 08h | WORD | data bits |
| 0Ah | WORD | stop bits |
| 0Ch | WORD | break status<br>0000h off |
| 0Eh | WORD | flow control protocol |
| 10h | BYTE | input block |
| 11h | BYTE | output block |
| 12h | WORD | low threshold |
| 14h | WORD | high threshold |
| 16h | WORD | segment of buffer |
| 18h | WORD | offset of buffer |
| 1Ah | WORD | input buffer length |
| 1Ch | WORD | output buffer length |
| 1Eh | BYTE | auxiliary address |
| 1Fh | BYTE | spare |
| 20h | 4 WORDs | spares |

---

### INT 2Fh Function D3CBh

*LapLink Quick Connect v6*
*(undocumented)*

***API***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = D3CBh | CX = 534Bh (except function 0002h) |
| CX = function | |
| 0002h get *unknown values* | BX:AX -> *unknown*<br>CL = *unknown*<br>CH = *unknown*<br>DX = *unknown*<br>DI = *COM1 I/O port*<br>SI = *COM2 I/O port* |
| 0003h *initialization* | |
| 0004h *unknown* | |
| 0005h *initialization* | |
| 0006h reset/clear *unknown value* | AX = 0000h<br>ES:DI -> next byte after *unknown value* cleared by this call |
| 0007h *initialization* | |
| 0008h uninstall | BX = status<br>0000h successful<br>FFFFh incomplete, stub remains in memory |

**Conflicts:** none, but see Table 2-2

---

### INT 65h

*EZRECV v1.0*
*(undocumented)*

***API***

| **Registers at call:** | **Return Registers:** |
|---|---|
| | BH = COM port being used<br>BL = *speed*<br>CH = *unknown*<br>CL = *unknown*<br>DX = *unknown*<br>DS = *unknown*<br>ES = EZRECV data segment |
| AX = function | |
| 0000h *unknown* | AX = *unknown* or FFFFh |
| 0001h *unknown* | AX = status (0000h or 0001h) |
| 0002h *unknown* | AX = status (0000h or 0001h) |
| *0003h set unknown variable* to 0001h | AX = 0000h |
| 0004h *unknown* | AX = *unknown* |

**Program:** EZRECV is a background Zmodem file receiver by Express Consulting.
**Conflicts:** see Table 2-3

### INT 7Fh Function 01h
*TIGA Communications Driver v2.05*

***GET ENTRY POINTS***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 01h | BX = 1234h if installed<br>DX:AX -> array of FAR entry points<br>CH = driver major version<br>CL = driver minor version |

**Conflicts:** see Table 2-4
**Details:** TIGACD 2.05 returns CF set on unrecognized functions in AX.
**See also:** Functions 1234h and 4321h

### INT 7Fh Function 1234h
*TIGA Communications Driver v2.05*

***UNINSTALL***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1234h | none |

**Conflicts:** see Table 2-4
**See also:** Function 4321h

### INT 7Fh Function 2525h
*TIGA Communications Driver v2.05*

***UNDETERMINED FUNCTION***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 2525h<br>BX = *unknown* | *unknown* |

**Conflicts:** see Table 2-4
**See also:** Functions 4321h and 5555h

### INT 7Fh Function 4321h
*TIGA Communications Driver v2.05*

***INSTALLATION CHECK***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4321h | AX = 0000h if installed |

**Details:** INT 7Fh is the default, but may be overridden.
**Conflicts:** see Table 2-4
**See also:** Functions 01h, 1234h, 2525h, 4321h, and 5555h

### INT 7Fh Function 5555h
*TIGA Communications Driver v2.05*

***UNDETERMINED FUNCTION***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5555h<br>BX = *unknown* | *unknown* |

**Conflicts:** see Table 2-4
**See also:** Function 4321h

### INT F1h
*AQUEDUCT, PIPELINE*

***GET DATA AREA ADDRESS***

| **Registers at call:** | **Return Registers:** |
|---|---|
| none | AX:BX -> data area |

**Program:** AQUEDUCT and PIPELINE are TSRs by James W. Birdsall to connect COM1 and COM2 in software
**Details:** The installation check consists of testing for the following signature immediately preceding the interrupt handler: "JWBtvv" where 't' is either "A" for AQUEDUCT or "P" for PIPELINE and "vv" is a two-digit version number

### INT FDh
*TFPCX*

### *INSTALLATION CHECK*

**Registers at call:** 
AH = function (see separate entries below)

**Return Registers:** 
vary

**Program:** TFPCX is an interface between modem and terminal program for packet-radio communications.

**Details:** The installation check consists of testing for the string "N5NX" three bytes beyond the interrupt handler; INT FDh is the default, but may be changed, so the full installation check consists of scanning for the signature. TFPCX returns AX=FFFFh on any unsupported function call.

**Conflicts:** DoubleDOS (Chapter 49)

**See also:** Functions 01h, 03h, and FEh

### INT FDh Function 01h
*TFPCX*

### *TEST FOR CHARACTER WAITING*

**Registers at call:** 
AH = 01h

**Return Registers:** 
AX = status 
0000h no characters waiting 
0001h character available for input

**Conflicts:** DoubleDOS (Chapter 49)

**See also:** Function 02h

### INT FDh Function 02h
*TFPCX*

### *GET CHARACTER*

**Registers at call:** 
AH = 02h

**Return Registers:** 
AL = character

**Details:** This call is only allowed if Function 01h indicated that a character is available.

All available characters should be read before sending any additional characters.

**Conflicts:** DoubleDOS (Chapter 49)

**See also:** Function 01h,Function 03h

### INT FDh Function 03h
*TFPCX*

### *OUTPUT CHARACTER*

**Registers at call:** 
AH = 03h 
AL = character to send

**Return Registers:** 
none

**Conflicts:** DoubleDOS (Chapter 49)

**See also:** Function 02h

### INT FDh Function FEh
*TFPCX*

### *GET VERSION*

**Registers at call:** 
AH = FEh

**Return Registers:** 
AH = major version 
AL = minor version

**Program:** TFPCX is an interface between modem and terminal program for packet-radio communications.

**Conflicts:** DoubleDOS (Chapter 49)

**See also:** Functions 01h and 03h

## CHAPTER 13

# Keyboard Enhancers

## Anarkey

ANARKEY.COM is a commandline recall program by Steven Calwas.

---

### INT 2Fh Function E300h
### *INSTALLATION CHECK*

*ANARKEY.COM*

**Registers at call:**
AX = E300h

**Return Registers:**
AL = status
- 00h not installed
- FEh installed but suspended (v3.0+)
- FFh installed

**Details:** E3h is the default function number, but it may be set to any value from C0h to FFh.
**Conflicts:** none, but see Table 2-2
**See also:** Functions E100h and E301h to E307h, Newkey INT 66h (below)

---

### INT 2Fh Function E301h
### *GET UNKNOWN VALUE*

*ANARKEY.COM v2+*
*(undocumented)*

**Registers at call:**
AX = E301h

**Return Registers:**
DX:BX -> *unknown*

**Conflicts:** none, but see Table 2-2
**See also:** Function E300h

**13-1 Format of returned data structure for ANARKEY v2.0:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -7 | 7 BYTEs | signature ('ANARKEY') |
| 00h | WORD | *unknown* (0001h in v2.0) |
| 02h | WORD | *unknown* (0001h in v2.0) |
| 04h | WORD | *unknown* (0000h in v2.0) |
| 06h | WORD | PSP segment of next program loaded |

**13-2 Format of returned data structure for ANARKEY v3+:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -1 | BYTE | multiplex number |
| 00h | WORD | *unknown* (0001h in v3.0-4.0) |
| 02h | WORD | *unknown* (0001h in v3.0-4.0) |
| 04h | BYTE | *unknown* (0000h in v3.0-4.0) |
| 05h | WORD | PSP segment of next program loaded |

---

### INT 2Fh Function E302h
### *UNDETERMINED FUNCTION*

*ANARKEY.COM v3+*
*(undocumented)*

**Registers at call:**
AX = E302h
BL = *unknown*

**Return Registers:**
*unknown*

**Conflicts:** none, but see Table 2-2
**See also:** Function E300h

---

### INT 2Fh Function E303h
### *ANARKMD API*

*ANARKEY.COM v3+*
*(undocumented)*

**Registers at call:**
AX = E303h
BL = function
- 01h toggle insert mode
- 02h display contents of history buffer
- 03h write history buffer to file
  - ES:DX -> file name
- 04h clear history buffer
- 05h undefine all aliases
- 06h show aliases
- 07h list programs using Unix switchar
- 08h jump to bottom of history buffer
- 09h (v4.0) add string to history buffer
  - ES:DX -> ASCIZ string
- 0Ah (v4.0) *unknown*
  - ES:DX -> *unknown*
- 0Bh (v4.0) copy string to edit buffer for use as next input line
  - ES:DX -> ASCIZ string
- 0Ch (v4.0) *unknown*
- 0Dh (v4.0) copy *unknown value* to *unknown location*
- 0Eh (v4.0) *unknown*
- 0Fh (v4.0) *unknown*
- 10h (v4.0) set *unknown* flag
- 11h (v4.0) display error message about running in EMS under Windows

**Return Registers:**
*unknown*

**Conflicts:** none, but see Table 2-2
**See also:** Function E300h

---

### INT 2Fh Function E304h
### *UNDETERMINED FUNCTION*

*ANARKEY.COM v2+*
*(undocumented)*

**Registers at call:**
AX = E304h
BL = *unknown*

**Return Registers:**
*unknown*

**Conflicts:** none, but see Table 2-2
**See also:** Function E300h

---

### INT 2Fh Function E305h
### *ENABLE/SUSPEND ANARKEY*

*ANARKEY.COM v3+*
*(undocumented)*

**Registers at call:**
AX = E305h
BL = new state
- 01h suspended
- 00h enabled

**Return Registers:**
none

**Conflicts:** none, but see Table 2-2
**See also:** Function E300h

---

**INT 2Fh Function E306h** *ANARKEY.COM v4.0 (undocumented)*
***GET UNKNOWN VALUE***

**Registers at call:**
AX = E306h

**Return Registers:**
AX = *unknown*

**Conflicts:** none, but see Table 2-2
**See also:** Function E300h

---

**INT 2Fh Function E307h** *ANARKEY.COM v4.0 (undocumented)*
***GET UNKNOWN VALUE***

**Registers at call:**
AX = E307h

**Return Registers:**
AX = *unknown*
BL = *unknown*

**Conflicts:** none, but see Table 2-2
**See also:** Function E300h

## DOSKEY

**INT 2Fh Function 4800h** *DOS 5+ DOSKEY*
***INSTALLATION CHECK***

**Registers at call:**
AX = 4800h

**Return Registers:**
AL = nonzero if installed (DOS 5.0 and 6.0 return AX=AA02h)
ES = segment of DOSKEY resident portion

**Details:** DOSKEY chains the interrupt if AL is not 00h or 10h on entry.
**Conflicts:** none, but see Table 2-2
**See also:** Function 4810h, PCED Function 4800h (below)

---

**INT 2Fh Function 4810h** *DOS 5+ DOSKEY, PCED v2.1*
***READ INPUT LINE FROM CONSOLE***

**Registers at call:**
AX = 4810h
DS:DX -> line buffer (see Table 13-3)

**Return Registers:**
AX = 0000h if successful

**Details:** The first byte (length) of the buffer MUST be 80h, or DOSKEY chains to the previous handler; PCED allows sizes other than 80h.

If the user's input is a macro name, no text is placed in the buffer even though AX=0000h on return; the program must immediately issue this call again to retrieve the expansion of the macro. Similarly, if the user enters a special parameter such as $*, this call must be repeated to retrieve the expansion; on the second call, DOSKEY overwrites the macro name on the screen with its expansion.

Unlike DOSKEY, PCED expands all macros on the first call, so it is not necessary to make two calls; since the buffer is not empty on return, DOSKEY-aware programs will not make the second call. DOSKEY chains the interrupt if AL is not 00h or 10h on entry.

**Conflicts:** none, but see Table 2-2
**See also:** Function 4800h, INT 21h Function 0Ah (Chapter 24)

**13-3 Format of line buffer**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | maximum characters buffer can hold |
| 01h | BYTE | (call) number of chars from last input which may be recalled<br>(return) number of characters actually read, excluding CR |
| 02h | N BYTEs | actual characters read, including the final carriage return |

## PCED

PCED v2.1 is a command line editor/history/macro facility by Cove Software. It is the commercial version of the freeware CED.

---

### INT 2Fh Function 4800h
*PCED v2.1*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = 4800h | AX = AACDh if installed<br>ES = segment of PCED kernel (PCED has multiple code segments) |

**Details:** DOSKEY also responds to this call if installed, returning AX=AA02h.

Unlike DOSKEY, PCED does *not* chain if AL contains an unsupported function code. It IRETs with all registers intact.

**Conflicts:** none, but see Table 2-2

**See also:** DOSKEY Function 4800h (above)

---

### INT 2Fh Function 48C0h
*PCED v2.1*

***PCED API***

| Registers at call: | Return Registers: |
|---|---|
| AX = 48C0h | CF clear if successful |
| DX = API function code | CF set on error |
| other registers as required by the specified function | AX = PCED error code<br>other registers as appropriate for API function |

**Details:** The full API information is available from Cove Software.

**See also:** PCED Function 4800h, Functions 48C1h to 48C3h

---

### INT 2Fh Function 48C1h
*PCED/VSTACK*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = 48C1h | AX = 0000h if installed |
| BL = 00h | BX = VSTACK resident segment |

**Program:** VSTACK is a resident backscroll utility included as part of the PCED package.

**Details:** VSTACK chains the interrupt if BL is nonzero on entry.

**See also:** Functions 48C0h and 48C2h

---

### INT 2Fh Function 48C2h
*PCED/ATTRIB*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = 48C2h | AX = 0000h if installed |
| BL = 00h | BX = ATTRIB resident segment |

**Program:** ATTRIB is a resident file attribute changer included as part of the PCED package.

**Details:** PCED/ATTRIB chains the interrupt if BL nonzero on entry.

**See also:** Functions 48C0h, 48C1h, and 48C3h

---

### INT 2Fh Function 48C3h
*PCED/KEYDEF*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = 48C3h | AX = 0000h if installed |
| BL = 00h | BX = KEYDEF resident segment |

**Program:** KEYDEF is a resident keyboard redefinition utility included as part of the PCED package.

**Details:** PCED/KEYBUF chains the interrupt if BL is nonzero on entry.

**See also:** Functions 48C0h, 48C2h, and 48C4h

### INT 2Fh Function 48C4h *PCED/FLIST*
### *INSTALLATION CHECK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 48C4h | AX = 0000h if installed |
| BL = 00h | BX = FLIST resident segment |

**Program:** FLIST is a resident filelist processor included as part of the PCED package

**Details:** PCED/FLIST chains the interrupt if BL is nonzero on entry.

**See also:** Functions 48C0h, 48C3h, and 48C5h

### INT 2Fh Function 48C4h, Subfunction 00h *PCED/ASSOC*
### *INSTALLATION CHECK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 48C4h | AX = 0000h if installed |
| BL = 00h | BX = ASSOC resident segment |

**Program:** ASSOC is a resident utility included as part of the PCED package which associated files with executable programs based on their extensions.

**Details:** ASSOC chains the interrupt if BL > 02h on entry.

**See also:** Functions 48C0h and 48C4h, Function 48C5h Subfunctions 01h and 02h

### INT 2Fh Function 48C5h, Subfunction 01h *PCED/ASSOC*
### *VERSION CHECK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 48C5h | AX = 0000h if installed |
| BL = 01h | BX = binary ASSOC version (BL = major, BH = minor) |

**Details:** ASSOC chains the interrupt if BL > 02h on entry.

**See also:** Function 48C0h, Function 48C5h Subfunctions 00h and 02h

### INT 2Fh Function 48C5h, Subfunction 02h *PCED/ASSOC*
### *ASSOCIATION TEST*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 48C5h | AX = status |
| BL = 02h | 0000h if filename is unknown |
| DS:SI -> ASCIZ filename | 0001h if there is an association defined for the file |
| | BX destroyed |

**Details:** ASSOC chains the interrupt if BL > 02h on entry.

**See also:** Function 48C0h, Function 48C5h Subfunctions 00h and 01h

## Phantom of the Keyboard

Phantom of the Keyboard II is a shareware keystroke recorder/replayer by P2 Enterprises.

### INT 2Fh Function E100h *Phantom2 v1.1+*
### *INSTALLATION CHECK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = E100h | AX = 0001h if installed |
| | DS:SI -> ASCIZ hotkey name |
| | DS:DI -> ASCIZ recording filename |

**Conflicts:** none, but see Table 2-2

**See also:** Functions E101h, E102h, and E103h; ANARKEY Function E300h (above)

### INT 2Fh Function E101h
*Phantom2 v1.1+*

#### *FUNCTION REQUEST*

**Registers at call:**
AX = E101h
BX = function mask (see Table 13-4)
CX = code for hotkey (as returned by INT 16h Function 00h) if BX bit 6 set
DS:DX -> ASCIZ filespec if BX bit 7 set

**Return Registers:**
none

**Conflicts:** none, but see Table 2-2
**See also:** Function E100h

**13-4 Bitfields for function mask**

| | |
|---|---|
| bit 0 | record |
| bit 1 | play |
| bit 2 | QuickPlay |
| bit 3 | loop |
| bit 4 | mode display toggle |
| bit 5 | sound toggle |
| bit 6 | set hotkey |
| bit 7 | set filespec |

### INT 2Fh Function E102h
*Phantom2 v1.1+*

#### *UNINSTALL*

**Registers at call:**
AX = E102h

**Return Registers:**
AX = status
0001h removal successful
0002h not installed as TSR
FFFFh disabled but not removed

**Conflicts:** none, but see Table 2-2
**See also:** Function E100h

### INT 2Fh Function E103h
*Phantom2 v2.8*

#### *SET UNKNOWN VALUE FLAG*

**Registers at call:**
AX = E103h

**Return Registers:**
AX = 0001h

**Conflicts:** none, but see Table 2-2
**See also:** Function E100h

## Miscellaneous

### INT 10h Function FAh
*FASTBUFF.COM*

#### *INSTALLATION CHECK*

**Registers at call:**
AH = FAh

**Return Registers:**
AX = 00FAh if installed
ES = segment of resident code

**Program:** FASTBUFF.COM is a keyboard speedup/screen blanking utility by David Steiner
**Conflicts:** EGA Register Interface Library (Chapter 8)

### INT 12h Function 1806h
*KEYBUI v2.0+*

#### *INSTALLATION CHECK*

**Registers at call:**
CX = 1806h

**Return Registers:**
AX = kilobytes of contiguous memory starting at absolute address 00000h
CX = 1960h if installed

**Program:** KEYBUI is a resident keyboard driver by Johan Zwiekhorst which allows accented characters and box drawing on standard QWERTY keyboards; it also provides break-to-DOS and screen blanking capabilities.
**Conflicts:** BIOS INT 12h (Chapter 4), Borland C++ IDE (Chapter 54)

---

### INT 14h Function 82h
*KEYBOARD FOSSIL*

***API***

| Registers at call: | Return Registers: |
|---|---|
| AH = 82h | none |

**Conflicts:** COURIERS (Chapter 12)
**See also:** Function 7Eh

---

### INT 16h Function 4252h
*TEXTCAP 2.0*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = 4252h | AX = 5242h if installed |

**Program:** TEXTCAP 2.0 is a heavily modified (by Gisbert W. Selke) version of the *PC* Magazine utility CAPTURE written by Tom Kihlken.
**See also:** Functions 4253h and 4254h

---

### INT 16h Function 4253h
*TEXTCAP 2.0*

***UNINSTALL***

| Registers at call: | Return Registers: |
|---|---|
| AX = 4253h | AX = segment of resident code |

**Details:** The uninstall code does not check whether interrupt vectors have been chained by other programs.

The caller must free the main memory block (using the returned segment).
**See also:** Functions 4252h and 4254h

---

### INT 16h Function 4254h
*TEXTCAP 2.0*

***DUMP TEXT SCREEN TO FILE***

| Registers at call: | Return Registers: |
|---|---|
| AX = 4254h | AX = status<br>4254h if screen dump will be written as soon as disk becomes idle<br>5442h if screen dump written |

**See also:** Functions 4252h and 4253h

---

### INT 16h Function 4D4Fh
*M16_KBD.COM v5.6*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = 4D4Fh | AX = 6F6Dh if installed<br>ES = segment of resident code |

**Program:** M16_KBD is a shareware Cyrillic keyboard driver by I.V. Morozov
**See also:** INT 10h Function 1130h Subfunction 4D4Fh (Chapter 8)

---

### INT 16h Function 70h
*FAKEY.COM*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AH = 70h | AX = 1954h if installed |

**Program:** FAKEY is a keystroke faking utility by System Enhancement Associates.
**See also:** Functions 71h to 73h

### INT 16h Function 71h

FAKEY.COM

*PUSH KEYSTROKES*

**Registers at call:**
AH = 71h
CX = number of keystrokes
DS:SI -> array of words containing keystrokes to be returned by Function 00h

**Return Registers:**
none

**See also:** Functions 70h and 72h, standard Functions 00h and 05h (Chapter 4)

### INT 16h Function 72h

FAKEY.COM

*CLEAR FAKED KEYSTROKES*

**Registers at call:**
AH = 72h

**Return Registers:**
none

**See also:** Functions 70h and 71h

### INT 16h Function 73h

FAKEY.COM

*PLAY TONES*

**Registers at call:**
AH = 73h
CX = number of tones to play
DS:SI -> array of tones (see Table 13-5)

**Return Registers:**
none

**Program:** FAKEY is a keystroke faking utility by System Enhancement Associates.
**See also:** INT 15h Function 1019h (Chapter 44)

13-5 Format of tone array entries:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | divisor for timer channel 2 |
| 02h | WORD | duration in clock ticks |

### INT 16h Function CAh

STACKEY.COM v3.00

*API*

**Registers at call:**
AH = CAh
BX = 736Bh ("sk")
CX = 736Bh
AL = function
- 00h installation check
- 01h place keystroke in buffer
  - DX = keystroke (DH = scan code, DL = ASCII character)
- 02h flush STACKEY and BIOS keyboard buffers

**Return Registers:**
AX = CAFFh if installed
- BX = segment of resident code
- CX = STACKEY version (CH = major, CL = minor)

DX = words available in keyboard buffer (00h)

DX = words available in keyboard buffer (FFFFh on error) (01h)

**Program:** STACKEY is a shareware keyboard-input faking TSR by CtrlAlt Associates.
**See also:** EGAPAL/ VGAPAL Function CA00h (Chapter 8)

### INT 16h Function FFh

KBUF

*ADD KEY TO TAIL OF KEYBOARD BUFFER*

**Registers at call:**
AH = FFh
DX = scan code

**Return Registers:**
AL = status
- 00h success
- 01h failure

**Program:** KBUF is a keyboard buffer expander by Mark Adler.

**Conflicts:** OPTIMA Zoom TSR (Chapter 9), PC Cache (Chapter 17), CPTASK (Chapter 49), PC Tools (Chapter 61), DRIVEMAP (NI)
**See also:** Standard Function 05h (Chapter 4)

---

### INT 21h Function 0A00h
*WCED v1.6+*

### *INSTALLATION CHECK*

**Registers at call:**
AX = 0A00h
DS:DX -> 6-byte buffer whose first two bytes must be 00h

**Return Registers:**
buffer offset 02h-05h filled with "Wced" if installed

**Program:** WCED is a free command-line editor and history utility by Stuart Russell.
**Conflicts:** MS-DOS Function 0Ah (Chapter 24)
**See also:** CED Function FFh

---

### INT 21h Function FFh
*CED (Command EDitor)*

### *INSTALLABLE COMMANDS*

**Registers at call:**
AH = FFh
AL = subfunction
    00h add installable command
        BL = mode
            bit 0 = 1 callable from DOS prompt
            bit 1 = 1 callable from application DS:SI -> CR-terminated command name ES:DI -> FAR routine entry point
    01h remove installable command
        DS:SI -> CR-terminated command name
    02h reserved, may be used to test for CED installation

**Return Registers:**
CF clear if successful
CF set on error
    AX = status
        01h invalid function
        02h command not found (subfunction 01h only
        08h insufficient memory (subfunction 00h only)
        0Eh bad data (subfunction 00h only)
AH = FFh if CED not installed

**Program:** CED is a shareware DOS command-line enhancer by Christopher J. Dunford.
**Conflicts:** DOSED, DOS OEM Function (Chapter 24), DOS/4GW (Chapter 35), GO32 (Chapter 37), Viruses (Chapter 59), FLU_SHOT+ (Chapter 59), Topware (NI)
**See also:** WCED Function 0A00h

---

### INT 21h Function FFh
*DOSED.COM*

### *INSTALLATION CHECK*

**Registers at call:**
AH = FFh
DS:SI -> "DOSED"
ES = 0000h

**Return Registers:**
ES:DI -> "DOSED" if installed

**Program:** DOSED is a free DOS commandline editor/ history buffer by Sverre H. Huseby.
**Conflicts:** CED, DOS OEM Function (Chapter 24), DOS/ 4GW (Chapter 35), GO32 (Chapter 37), Viruses (Chapter 59), FLU_SHOT+ (Chapter 59), Topware (NI)

---

### INT 2Fh Function CF00h
*TEMPLEXX 1.0*

### *INSTALLATION CHECK*

**Registers at call:**
AX = CF00h

**Return Registers:**
AL = FFh if installed

**Program:** TEMPLEXX is a popup keyboard template by Henson Scientific, Inc.
**Details:** Values in AL other than 00h cause an immediate return without modifying any registers.
**Conflicts:** none, but see Table 2-2

### INT 2Fh Function E000h
*StuffIt v3.21+*

*INSTALLATION CHECK*

**Registers at call:**
AX = E000h
DX = 5354h ("ST")

**Return Registers:**
AL = FFh if installed
BX = version (BH = major, BL = BCD minor)
DX = segment of resident code

**Program:** StuffIt is a freeware delayed keyboard stuffer by Terje Mathisen.
**Conflicts:** SETDRVER (Chapter 66); see also Table 2-2

### INT 2Fh Function F000h
*4MAP (undocumented)*

*INSTALLATION CHECK*

**Registers at call:**
AX = F000h

**Return Registers:**
AX = 00FFh

**Program:** 4MAP is a keybinding program for 4DOS (see Chapter 28) by Ho-Ping Tseng.
**Details:** 4MAP returns AX=00FFh for any value of AL not listed here.
**Conflicts:** none, but see Table 2-2
**See also:** Functions F001h and F002h, 4DOS Function D44Dh (Chapter 28)

### INT 2Fh Function F001h
*4MAP (undocumented)*

*GET KEY MAPPINGS*

**Registers at call:**
AX = F001h

**Return Registers:**
ES:BX -> key mappings

**Conflicts:** none, but see Table 2-2
**See also:** Function F000h

### INT 2Fh Function F002h
*4MAP (undocumented)*

*INSERT CHARACTER INTO UNKNOWN BUFFER*

**Registers at call:**
AX = F002h
BL = character to insert

**Return Registers:**
AX = status
0000h successful
0001h buffer full

**Conflicts:** none, but see Table 2-2
**See also:** Functions F000h and F003h

### INT 2Fh Function F003h
*4MAP (undocumented)*

*INSERT CHARACTER INTO UNKNOWN BUFFER*

**Registers at call:**
AX = F003h
BL = character to insert

**Return Registers:**
AX = status
0000h successful
0001h buffer full

**Program:** 4MAP is a keybinding program for 4DOS (see Chapter 28) by Ho-Ping Tseng.
**Conflicts:** none, but see Table 2-2
**See also:** Functions F000h and F002h

### INT 48h
*KEYBOARD*
*PCjr only*

*CORDLESS KEYBOARD TRANSLATION*

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Western Digital Super BIOS (Chapter 10), Compaq UILIB (Chapter 54), Watstar PC Network (NI)
**See also:** PCjr INT 49h

---

### INT 49h

*SYSTEM DATA*

### *NON-KEYBOARD SCAN-CODE TRANSLATION TABLE*

*PCjr only*

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Texas Instruments PC (Chapter 8), MAGic (Chapter 23), Watstar PC Network (NI)
**See also:** PCjr INT 48h

**13-6 Format of translation table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | number of nonkeyboard scancodes in the table |
| 01h | N WORDs | high byte 00h (NUL) byte scancode with low order byte representing the scancode mapped values relative to their input values within the range of 56h through 7Eh |

---

### INT 66h

*Newkey v5.4*

### *INSTALLATION VECTOR*

| **Registers at call:** | **Return Registers:** |
|---|---|
| none | immediately (IRET) |

**Program:** Newkey is a shareware keyboard macro program by Frank A. Bell.
**Details:** The installation check consists of testing for the signature bytes FDh FCh FFh FEh at offset 03h in the interrupt handlers segment; Newkey may use any interrupt from 60h through 67h and will install on the highest vector in this range which is unused (normally 66h)
**BUG:** The code obviously intends to use INT F0h-FE, INT 70h-77, and INT 68h-6F before falling back to INT 60h-67, but only uses the last of these ranges in v5.4.
**Conflicts:** see Table 2-3
**See also:** INT 2Fh Function E300h

---

### INT 79h

*AVATAR.SYS*

### *FAST GET KEYSTROKE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| none | CF set if no keystroke available<br>AX = FFFFh<br>CF clear if key pressed<br>AX = keystroke |

**Program:** AVATAR.SYS is a CON driver by George Adam Stanislav which interprets AVATAR command codes just as ANSI.SYS interprets ANSI commands.
**Details:** If a keystroke is available, it is removed from the keyboard buffer before being returned.
**Conflicts:** ADP-60 controller (Chapter 10), DBOS DOS Extender (Chapter 37), GO32 (Chapter 37), AutoCAD Device Interface (Chapter 66)
**See also:** INT 29h

## CHAPTER 14

# *Screen Blankers*

## After Dark

AD-DOS is the DOS version of the After Dark screen blanker for MS Windows.

AH=C0h as shown in this section is the default multiplex number; if it is already in use, After Dark will try successive values up to AH=FFh. As with all TSRs using multiplex numbers in this range, there are numerous potential conflicts, which are listed in Table 2-2.

---

### INT 2Fh Function C000h
*AD-DOS*

***INSTALLATION CHECK***

**Registers at call:**
AX = C000h

**Return Registers:**
AL = FFh if installed
BX = 4144h ('AD')
CX = 2D44h ('-D')
DX = 4F53h ('OS')

**See also:** Functions C001h, C003h, C005h, C007h, C009h, and C020h

---

### INT 2Fh Function C001h
*AD-DOS*

***GET RESIDENT CODE SEGMENT***

**Registers at call:**
AX = C001h
ES = 0000h

**Return Registers:**
AL = 00h if successful
ES = AD-DOS TSR Code Segment

**See also:** Function C000h

---

### INT 2Fh Function C002h
*AD-DOS*

***CHECK FOR NEW INPUT***

**Registers at call:**
AX = C002h

**Return Registers:**
AL = 00h if succesful
BX = status
0000h no input since last check
0001h new input available

**Details:** This call also resets the new-input flag.

**See also:** Functions C000h and C004h

---

### INT 2Fh Function C003h
*AD-DOS*

***SET MINUTES TO WAIT***

**Registers at call:**
AX = C003h
BX = minutes to wait before blanking screen

**Return Registers:**
AL = 00h if successful

**Details:** This call also resets the delay timer, so that the screen will not blank until at least the specified number of minutes after the call.
**Conflicts:** see Table 2-2
**See also:** Functions C000h and C004h

---

### INT 2Fh Function C004h
*AD-DOS*
### *GET MINUTES TO WAIT*

**Registers at call:**
AX = C004h

**Return Registers:**
AL = 00h if succesful
BX = minutes to wait before blanking screen

**See also:** Functions C000h and C003h

---

### INT 2Fh Function C005h
*AD-DOS*
### *SET BLANKER STATUS*

**Registers at call:**
AX = C005h
BX = new state (0000h inactive, 0001h active [default])

**Return Registers:**
AL = 00h if successful

**See also:** Function C006h

---

### INT 2Fh Function C006h
*AD-DOS*
### *GET BLANKER STATUS*

**Registers at call:**
AX = C006h

**Return Registers:**
AL = 00h if successful
BX = current state of screen blanker (0000h inactive, 0001h active)

**Conflicts:** see Table 2-2
**See also:** Functions C000h and C005h

---

### INT 2Fh Function C007h
*AD-DOS*
### *SET HOT KEY*

**Registers at call:**
AX = C007h
BX = hot key
CL = hot key shift status

**Return Registers:**
AL = 00h if successful

**See also:** Function C008h

---

### INT 2Fh Function C008h
*AD-DOS*
### *GET CURRENT HOT KEY*

**Registers at call:**
AX = C008h

**Return Registers:**
AX = status
- 0000h successful
  - BX = hot key
  - CL = hot key shift status
- 0008h otherwise

**See also:** Functions C000h and C007h

---

### INT 2Fh Function C009h
*AD-DOS*
### *UNBLANK MONITOR*

**Registers at call:**
AX = C009h

**Return Registers:**
AL = 00h if successful

**Details:** This function works by simulating keyboard activity.
**Conflicts:** see Table 2-2

---

### INT 2Fh Function C00Ah
*AD-DOS*
### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = C00Ah

**Return Registers:**
AX = status
0000h successful
BH = *unknown*
BL = *unknown*
000Ah failed

---

### INT 2Fh Function C00Bh
AD-DOS

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = C00Bh

**Return Registers:**
AX = status
0000h successful
000Bh failed

---

### INT 2Fh Function C00Ch
AD-DOS

### *SET UNKNOWN INTERNAL FLAG TO 01h*

**Registers at call:**
AX = C00Ch

**Return Registers:**
AX = status
0000h successful
000Ch failed

---

### INT 2Fh Function C00Eh
AD-DOS

### *SET PASSWORD STATUS*

**Registers at call:**
AX = C00Eh
BX = new state (0000h disabled, 0001h enabled)

**Return Registers:**
*unknown*

**Conflicts:** see Table 2-2
**See also:** Functions C000h and C00Fh

---

### INT 2Fh Function C00Fh
AD-DOS

### *GET PASSWORD STATUS*

**Registers at call:**
AX = C00Fh

**Return Registers:**
BX = current state (0000h disabled, 0001h enabled)

**See also:** Function C00Eh

---

### INT 2Fh Function C020h
AD-DOS

### *GET AND RESET VxD API STATUS*

**Registers at call:**
AX = C020h

**Return Registers:**
AL = 00h if successful
BX = VxD API Status
0000h no error
0001h error
0100h neither Windows 3.X enhanced mode nor Windows/386 2.x is running
0200h VM API entry point not found (VxD not installed)

**Details:** This call resets the VxD API Status to zero.
**Conflicts:** see Table 2-2
**See also:** Windows callout Functions 1602h and 1607h (Chapter 43)

## Blank-It

Blank-It is a resident screen blanker by Rhode Island Soft Systems, Inc.

---

**INT 11h Function 0225h, Subfunction 6900h** *Blank-It*

***INSTALLATION CHECK***

**Registers at call:**
AX = 0225h
BX = 6900h

**Return Registers:**
BL = 23h
ES:DI -> ASCIZ "BLNKIT"

**Conflicts:** Alignment Check (Chapter 3), BIOS INT 11h (Chapter 4), SDLP (Chapter 11)
**See also:** Function 0225h Subfunctions 6902h, 6908h, and 6909h

---

**INT 11h Function 0225h, Subfunction 6901h** *Blank-It*

***SET TIMEOUT FOR SCREEN BLANKING***

**Registers at call:**
AX = 0225h
BX = 6901h
CX = timeout in timer ticks (18.2/second) or 0000h to disable timeout
largest value is 59 minutes (FBACh or 64428)

**Return Registers:**
CF clear if successful
DI = 6A6Ah (possibly also 6A6Ah for all following functions)
AH = *unknown*
CF set on error
AL = error code

**Conflicts:** Alignment Check (Chapter 3), BIOS INT 11h (Chapter 4), SDLP (Chapter 11)
**See also:** Function 0225h Subfunctions 6900h and 6904h

---

**INT 11h Function 0225h, Subfunction 6902h** *Blank-It*

***ENABLE THE SOFTWARE***

**Registers at call:**
AX = 0225h
BX = 6902h

**Return Registers:**
CF clear if successful
CF set on error
AL = error code

**Conflicts:** Alignment Check (Chapter 3), BIOS INT 11h (Chapter 4), SDLP (Chapter 11)
**See also:** Function 0225h Subfunctions 6900h and 6903h

---

**INT 11h Function 0225h, Subfunction 6903h** *Blank-It*

***DISABLE THE SOFTWARE***

**Registers at call:**
AX = 0225h
BX = 6903h

**Return Registers:**
CF clear if successful
CF set on error
AL = error code

**Conflicts:** Alignment Check (Chapter 3), BIOS INT 11h (Chapter 4), SDLP (Chapter 11)
**See also:** Function 0225h Subfunctions 6900h and 6902h

---

**INT 11h Function 0225h, Subfunction 6904h** *Blank-It*

***GET BLANKING TIMEOUT***

**Registers at call:**
AX = 0225h
BX = 6904h

**Return Registers:**
CF clear if successful
BX = timeout (see BX=6901h)
CF set on error
AL = error code

**Conflicts:** Alignment Check (Chapter 3), BIOS INT 11h (Chapter 4), SDLP (Chapter 11)
**See also:** Function 0225h Subfunctions 6900h and 6901h

---

**INT 11h Function 0225h, Subfunction 6905h** *Blank-It*

***ENABLE WINDOWS COMPATIBILITY MODE***

**Registers at call:**
AX = 0225h
BX = 6905h

**Return Registers:**
CF clear if successful
CF set on error
AL = error code

**Conflicts:** Alignment Check (Chapter 3), BIOS INT 11h (Chapter 4), SDLP (Chapter 11)
**See also:** Function 0225h Subfunctions 6900h and 6906h

### INT 11h Function 0225h, Subfunction 6906h

*Blank-It*

***DISABLE WINDOWS COMPATIBILITY MODE***

**Registers at call:**
AX = 0225h
BX = 6906h

**Return Registers:**
CF clear if successful
CF set on error
AL = error code

**Conflicts:** Alignment Check (Chapter 3), BIOS INT 11h (Chapter 4), SDLP (Chapter 11)
**See also:** Function 0225h Subfunctions 6900h and 6905h

### INT 11h Function 0225h, Subfunction 6907h

*Blank-It*

***UNBLANK THE SCREEN***

**Registers at call:**
AX = 0225h
BX = 6907h

**Return Registers:**
CF clear if successful
CF set on error
AL = error code

**Conflicts:** Alignment Check (Chapter 3), BIOS INT 11h (Chapter 4), SDLP (Chapter 11)
**See also:** Function 0225h Subfunctions 6900h and 6908h

### INT 11h Function 0225h, Subfunction 6908h

*Blank-It*

***BLANK THE SCREEN***

**Registers at call:**
AX = 0225h
BX = 6908h

**Return Registers:**
CF clear if successful
CF set on error
AL = error code

**Conflicts:** Alignment Check (Chapter 3), BIOS INT 11h (Chapter 4), SDLP (Chapter 11)
**See also:** Function 0225h Subfunctions 6900h and 6907h

### INT 11h Function 0225h, Subfunction 6909h

*Blank-It*

***SET HOTKEY FOR MANUAL BLANKING***

**Registers at call:**
AX = 0225h
BX = 6909h
CL = key scan code (see Table 14-1)

**Return Registers:**
CF clear if successful
CF set on error
AL = error code

**Conflicts:** Alignment Check (Chapter 3), BIOS INT 11h (Chapter 4), SDLP (Chapter 11)
**See also:** Function 0225h Subfunction 6900h

**14-1 Values for key scan code:**

| | |
|---|---|
| 00h | No hot key |
| 1Dh | Left CTRL |
| 2Ah | Left Shift |
| 36h | Right Shift |
| 57h | F11 |
| 58h | F12 |

## Explosiv

Explosiv is a shareware animated screen blanker for DOS and Windows 3 by Reidar Gresseth and Chris Hook. AH=C0h as shown in this section is the default multiplex number, but it may be reconfigured to any value from C0h through C9h. As with all software using multiplex numbers in this range, there are numerous possible conflicts, which are listed in Table 2-2.

### INT 2Fh Function C050h
### *INSTALLATION CHECK*

*Explosiv 2.00*
*(undocumented)*

**Registers at call:**
AX = C050h

**Return Registers:**
AL = 51h if installed

**Conflicts:** see Table 2-2
**See also:** Function C052h

### INT 2Fh Function C052h
### *UNINSTALL*

*Explosiv 2.00*
*(undocumented)*

**Registers at call:**
AX = C052h
DX:BX = address to return to on successful uninstall

**Return Registers:**
at specified address if successful
AL = 53h on error

**Details:** The specified return address must have the segment of the caller's PSP.
**See also:** Function C050h

### INT 2Fh Function C054h
### *GET UNKNOWN VALUE*

*Explosiv 2.00*
*(undocumented)*

**Purpose:** The purpose of this function has not yet been determined.

**Registers at call:**
AX = C054h

**Return Registers:**
AL = *unknown*

**Details:** This function returns the value set by Function C057h.
**See also:** Function C057h

### INT 2Fh Function C055h
### *SET BLANKING INTERVAL*

*Explosiv 2.00*
*(undocumented)*

**Registers at call:**
AX = C055h
BX = new interval in clock ticks

**Return Registers:**
none

**Conflicts:** see Table 2-2

### INT 2Fh Function C056h
### *SET UNKNOWN VALUE*

*Explosiv 2.00*
*(undocumented)*

**Purpose:** The purpose of this function has not yet been determined.

**Registers at call:**
AX = C056h
BL = *unknown*

**Return Registers:**
none

### INT 2Fh Function C057h
### *SET UNKNOWN VALUE*

*Explosiv 2.00*
*(undocumented)*

**Purpose:** The purpose of this function has not yet been determined.

**Registers at call:**
AX = C057h
BL = *unknown*

**Return Registers:**
none

**See also:** Function C054h

### INT 2Fh Function C058h
### *SET UNKNOWN VALUE*

*Explosiv 2.00*
*(undocumented)*

**Purpose:** The purpose of this function has not yet been determined.

**Registers at call:**
AX = C058h
BL = *unknown*

**Return Registers:**
none

**Conflicts:** see Table 2-2

### INT 2Fh Function C059h
### *CLEAR UNKNOWN FLAG*

*Explosiv 2.00*
*(undocumented)*

**Purpose:** The purpose of this function has not yet been determined.

**Registers at call:**
AX = C059h

**Return Registers:**
none

**Details:** This function clears the flag set by Function C05Ah.

**See also:** Function C05Ah

---

### INT 2Fh Function C05Ah
### *SET UNKNOWN VALUE*

*Explosiv 2.00*
*(undocumented)*

**Purpose:** The purpose of this function has not yet been determined.

**Registers at call:**
AX = C05Ah
BL = *unknown*

**Return Registers:**
none

**Details:** This function sets a flag which is cleared by Function C059h, then stores BL.

**See also:** Functions C059h and C05Bh

---

### INT 2Fh Function C05Bh
### *SET UNKNOWN VALUE*

*Explosiv 2.00*
*(undocumented)*

**Purpose:** The purpose of this function has not yet been determined.

**Registers at call:**
AX = C05Bh
BL = *unknown*

**Return Registers:**
none

**See also:** Functions C05Ah and C05Ch

---

### INT 2Fh Function C05Ch
### *SET UNKNOWN VALUE*

*Explosiv 2.00*
*(undocumented)*

**Purpose:** The purpose of this function has not yet been determined.

**Registers at call:**
AX = C05Ch
BL = *unknown*

**Return Registers:**
none

**Conflicts:** see Table 2-2

**See also:** Function C05Bh

## Miscellaneous

### INT 2Dh Function 10h
### *GET STATE/CONTROL INFORMATION*

*Burnout Plus v3.00*

**Registers at call:**
AL = 10h
AH = AMIS multiplex number for Burnout Plus (see Chapter 50)

**Return Registers:**
AL = 01h
BX = Burnout Plus status (see Table 14-2)
CX = Record of features loaded (see Table 14-3)
ES:DI -> Burnout Plus control structure (see Table 14-4)

**Program:** Burnout Plus is a DOS screen saver from Cove Software.

**See also:** AMIS INT 2Dh (Chapter 50)

**14-2 Bitfields for status:**

| | |
|---|---|
| bit 0 | screen is blanked |
| bit 1 | MS Windows is active (Burnout Plus deactivated) |
| bits 2-15 | reserved |

**14-3 Bitfields for features loaded/features enabled:**

| | |
|---|---|
| bit 0 | mouse activity monitor |
| bit 1 | passkey support |
| bit 2 | password support |
| bit 3 | continuous clear |
| bit 4 | software blanking |
| bit 5 | video activity monitor |

| | |
|---|---|
| bit 6 | disk activity monitor |
| bit 7 | activating keystroke suppression |

**14-4 Format of Burnout Plus control structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | size of structure in bytes |
| 01h | WORD | Burnout Plus version |
| 03h | WORD | screen blanking reset count in clock ticks |
| 05h | WORD | current countdown value in clock ticks |
| 07h | BYTE | type of timeout specification |
| 08h | BYTE | instant-blank hotkey |
| 09h | WORD | extended status information (see Table 14-5)<br>the bits for password, passkey, and software blanking are ignored and cannot be enabled or disabled externally |
| 0Bh | WORD | features enabled (see Table 14-3) |

**Note:** all fields except the first two may be modified by external programs to affect the operation of Burnout Plus

**14-5 Bitfields for extended status information:**

| | |
|---|---|
| bit 0 | Burnout Plus disabled |
| bit 1 | force screen to blank on next clock tick |
| bit 2 | restore screen if currently blanked |
| bits 3-15 | reserved |

**Note:** bits 1 and 2 are automatically cleared by Burnout Plus after blanking or restoring the screen

---

### INT 2Fh Function 6400h
*SCRNSAV2.COM*

***INSTALLATION CHECK***

**Registers at call:**
AX = 6400h

**Return Registers:**
AL = status
00h not installed
FFh installed

**Program:** SCRNSAV2.COM is a screen saver for PS/2s with VGA by Alan Ballard.
**Conflicts:** none, but see Table 2-2
**See also:** INT 10h Function 5555h (Chapter 8)

---

### INT 2Fh Function 93h
*InnerMission v1.7+*

***INSTALLATION CHECK***

**Registers at call:**
AH = 93h
BX = CX = AX

**Return Registers:**
AL = result
01h if installed but BX or CX differ from AX
FFh if installed and BX=CX=AX on entry
BX = segment of resident code

**Conflicts:** none, but see Table 2-2
**Program:** InnerMission is a shareware graphical screen blanker by Kevin Stokes.

---

### INT 2Fh Function C000h
*VGAsave v1.93*

***INSTALLATION CHECK***

**Registers at call:**
AX = C000h

**Return Registers:**
AL = FFh if installed
BX = segment of resident code

**Program:** VGAsave is a freeware VGA-specific, mouse-aware screenblanker by Bill Javurek.
**Details:** VGAsave may use any AH value from C0h through FFh; the transient portion of VGAsave compares the first 38 bytes of the resident code (addressed through BX) against its own copy of the resident code to complete the installation check
**Conflicts:** see Table 2-2

## INT 2Fh Function 6400h

# CHAPTER 15

# *Printer Services and Print Spoolers*

## Standard BIOS

### INT 17h Function 00h
**PRINTER**

***WRITE CHARACTER***

**Registers at call:**
AH = 00h
AL = character to write
DX = printer number (00h-02h)

**Return Registers:**
AH = printer status (see Table 15-1)

**Conflicts:** LPTx installation checks (below)

**See also:** Function 02h, Jetstream Function F1h (below), AX PC Function 84h (Chapter 6), NEC INT 1Ah Function 11h (Chapter 6), PC Tools INT 16h Function FFE3h (Chapter 61)

**15-1 Bitfields for printer status:**

| | |
|---|---|
| bit 7 | not busy |
| bit 6 | acknowledge |
| bit 5 | out of paper |
| bit 4 | selected |
| bit 3 | I/O error |
| bits 2,1 | unused |
| bit 0 | timeout |

### INT 17h Function 01h
**PRINTER**

***INITIALIZE PORT***

**Registers at call:**
AH = 01h
DX = printer number (00h-02h)

**Return Registers:**
AH = printer status (see Table 15-1)

**Details:** Some printers report that they are ready immediately after initialization when they actually are not; a more reliable result may be obtained by calling Function 02h after a brief delay.

**Conflicts:** LPTx installation checks (below)

**See also:** Function 02h, NEC INT 1Ah Function 10h (Chapter 6), PC-MOS Function FFh (Chapter 49)

### INT 17h Function 02h
**PRINTER**

***GET STATUS***

**Registers at call:**
AH = 02h
DX = printer number (00h-02h)

**Return Registers:**
AH = printer status (see Table 15-1)

**Details:** PRINTFIX from MS-DOS 5.0 hooks this function and always returns AH=90h.

**Conflicts:** LPTx installation checks (below), INSET installation check (below)
**See also:** Function 01h, Jetstream Function F2h (below), NEC INT 1Ah Function 12h (Chapter 6)

## Frieze

---

### INT 10h Function 0Fh
*Frieze v7.41+*

***INSTALLATION CHECK***

**Registers at call:**
AH = 0Fh
SI = F123h
DI = 321Fh

**Return Registers:**
AH = number of character columns
AL = video mode (see Table 8-2)
BH = active page (see INT 10h Function 05h)
SI = DI = F345h if installed

**Details:** The values returned in AL, AH, and BH are the same ones that the standard INT 10h Function 0Fh returns, for compatibility.
**Conflicts:** Coprocessor Error (Chapter 3), Video Function 0Fh (Chapter 8)
**See also:** Function 4Bh

---

### INT 10h Function 4Bh
*FRIEZE*

***API***

**Registers at call:**
AH = 4Bh
CL = function
- 00h (v6.x-) print window
  - AL = mode
    - 00h character
    - 01h normal
    - 02h sideways
- 01h load window
  - ES:BX -> ASCIZ filename from which to read
- 02h save window
  - ES:BX -> ASCIZ filename to which to write
- 03h set print width
  - AL = width in 1/4 inches
- 04h set print height
  - AL = height in 1/4 inches
- 05h reserved
- 06h set left margin
  - AL = printout margin in 1/4 inches
- 07h set window size
  - ES:BX -> four-WORD structure with Xmin, Ymin, Xmax, Ymax
- 08h reserved
- 09h set patterns
  - ES:BX -> 16-BYTE vector of screen->printer color correspondnces
- 0Ah get patterns
  - ES:BX -> 16-BYTE buffer for color correspondences
- 0Bh set mode
  - AL = mode
- 0Ch (v7.41) *unknown*
  - AL = *unknown*
    - 00h *unknown* (calls original INT 05h)
    - else *unknown*
- 0Dh (v7.41) *unknown*

AL = *unknown*
00h *unknown* (calls original INT 05h)
else *unknown*

**Return Registers:**
AX = status (see Table 15-2)

0Eh (v7.41) *unknown*
*other registers unknown*
0Fh get window
ES:BX -> four-WORD buffer for Xmin, Ymin, Xmax, Ymax
10h set print options
ES:BX -> printer options in same format as FRIEZE cmdline
11h initialize
ES:BX -> three-WORD array from CARDS.DAT for HRes, VRes, code
12h (v7.41) *unknown*
*other registers unknown*
13h (v7.41) *unknown*
*other registers unknown*

**Registers at call:**
AH = 4Bh
CL = function
14h get version
15h set parameters
ES:BX -> parameter table (see Table 15-3)
16h get parameters
ES:BX -> buffer for parameter table (see Table 15-3)
17h get printer resolution
ES:BX -> 12-WORD table for six horizontal/vertical resol pairs
18h (v8.0 only) reserved
50h (v7.41) get *unknown value*
51h (v7.41) get *unknown value*

**Return Registers:**
AH = major version (00h if FRIEZE version before 7)
AL = minor version
AX = status (see Table 15-2)
AX = status (see Table 15-2)
AX = status (see Table 15-2)
AX = *unknown*
ES = *unknown* (2348h)
AX = *unknown* (8432h)

**Conflicts:** Coprocessor Error (Chapter 3), Hercules GRAFIX (Chapter 8)
**See also:** Function 0Fh Subfunction F123h

**15-2 Values for status:**

| | |
|---|---|
| 00h | successful |
| 01h | user aborted printout with ESC |
| 02h | reserved |
| 03h | file read error |
| 04h | file write error or printer error |
| 05h | file not found |
| 06h | invalid header (not an image or wrong screen mode) or can’t create file |
| 07h | file close error |
| 08h | disk error |
| 09h | (v7.0+) printer error |
| 0Ah | invalid function |
| 0Bh | (v7.0+) can’t create file |
| 0Ch | (v7.0+) wrong video mode |

**15-3 Format of parameter table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | top margin (1/100 inch) |
| 02h | WORD | left margin (1/100 inch) |
| 04h | WORD | horizontal size (1/100 inch) |
| 06h | WORD | vertical size (1/100 inch) |
| 08h | WORD | quality/draft mode<br>00h draft mode<br>01h quality mode<br>02h use horizontal/vertical resolution for output resolution |
| 0Ah | WORD | printer horizontal resolution (dots per inch) |
| 0Ch | WORD | printer vertical resolution (dots per inch) |
| 0Eh | WORD | reserved (FFFFh) |

**Note:** Any field which should remain unchanged may be filled with FFFFh.

## INSET

INSET is a text/graphics integration program.

---

### INT 17h Function 02h, Subfunction 0000h

*INSET*

***INSTALLATION CHECK***

**Registers at call:**
AH = 02h
DX = 0000h
CX = 07C3h (1987d)

**Return Registers:**
CX = 07C2h (1986d) if installed

**Conflicts:** Standard INT 17h (above)

---

### INT 17h Function CD00h

*INSET*

***EXECUTE COMMAND STRING***

**Registers at call:**
AX = CD00h
DS:DX -> ASCIZ command string
(max 80 bytes)

**Return Registers:**
CX = 07C2h (1986d)

**Details:** The user interface menus pop up after the last command, unless that command exits INSET.

---

### INT 17h Function CD01h

*INSET*

***GET IMAGE SIZE***

**Registers at call:**
AX = CD01h
DS:DX -> ASCIZ name of image file

**Return Registers:**
AX = height in 1/720th inch
BX = width in 1/720th inch
CX = 07C2h (1986d)

---

### INT 17h Function CD02h

*INSET*

***INITIALIZE***

**Registers at call:**
AX = CD02h

**Return Registers:**
CX = 07C2h (1986d)

**Details:** All open files are closed and the printer is reset.
**See also:** Function CD04h

### INT 17h Function CD03h

*INSET*

***EXECUTE INSET MENU WITHIN OVERRIDE MODE***

**Registers at call:**
AX = CD03h

**Return Registers:**
CX = 07C2h (1986d)

### INT 17h Function CD04h

*INSET*

***INITIALIZE LINKED MODE***

**Registers at call:**
AX = CD04h
ES:SI -> FAR routine for linked mode (see Table 15-4)

**Return Registers:**
CX = 07C2h

**See also:** Functions CD02h and CD08h

**15-4 Linked-mode routine calling convention:**

**Registers at call:**
AL = operation
- 00h send character in BL to printer
- 01h send CX bytes from DS:DX to printer
- 02h move print head to horizontal starting position of image

**Return Registers:**
AX = status
- 0000h success
- 0001h failure

### INT 17h Function CD05h

*INSET*

***START MERGING IMAGE INTO TEXT***

**Registers at call:**
AX = CD05h
DS:DX -> ASCIZ name of PIX file
CX = left margin of text in 1/720th inch

**Return Registers:**
AH = printer type
- 00h page-oriented (multiple images may be placed side-by-side)
- 01h line-oriented (use Function CD06h for vertical paper movement)

CX = 07C2h (1986d)

**See also:** Function CD07h

### INT 17h Function CD06h

*INSET*

***GRAPHICS LINE FEED***

**Registers at call:**
AX = CD06h

**Return Registers:**
AH = completion status
- 00h image complete
- 01h image incomplete

CX = 07C2h (1986d)

**See also:** Function CD09h

### INT 17h Function CD07h

*INSET*

***FLUSH GRAPHICS FROM MERGE BUFFER***

**Registers at call:**
AX = CD07h

**Return Registers:**
CX = 07C2h

**See also:** Function CD05h

### INT 17h Function CD08h

*INSET*

***CANCEL LINKED MODE***

**Registers at call:**
AX = CD08h

**Return Registers:**
CX = 07C2h

**See also:** Function CD04h

---

### INT 17h Function CD09h
### *ALTER TEXT LINE SPACING*

*INSET*

**Registers at call:**
AX = CD09h
CX = line spacing in 1/720th inch

**Return Registers:**
CX = 07C2h

**Details:** This function was not yet implemented as of the version described here; line spacing was fixed at 1/6 inch.

**See also:** Function CD06h

---

### INT 17h Function CD0Ah
### *GET SETUP*

*INSET*

**Registers at call:**
AX = CD0Ah
DS:DX -> buffer for IN.SET data

**Return Registers:**
CX = 07C2h

---

### INT 17h Function CD0Bh
### *START GETTING SCALED IMAGE*

*INSET*

**Registers at call:**
AX = CD0Bh
DS:SI -> ASCIZ pathname of .PIX file
BX = number of bitplanes
CX = number of rows in output bitmap
DX = number of columns in output bitmap

**Return Registers:**
AX = status
  0000h OK
  FFFFh error

**Details:** The image is returned in strips by repeated calls to Function CD0Ch.

---

### INT 17h Function CD0Ch
### *GET NEXT IMAGE STRIP*

*INSET*

**Registers at call:**
AX = CD0Ch

**Return Registers:**
AX = status
  0000h OK but not complete
  0001h OK and image complete
  FFFFh error
DS:SI -> buffer (max 4K) for bit map strip
CX = start row
DX = number of rows
BX = offset in bytes between bit planes

**Details:** The buffer may be overwritten by subsequent calls.

**See also:** Function CD0Bh

## NorthNet Jetstream

The NorthNet Jetstream is a high-performance DMA-driven parallel card able to drive printers at up to 80000 characters per second.

---

### INT 17h Function F0h
### *INSTALLATION CHECK*

*NorthNet Jetstream*

**Registers at call:**
AH = F0h
DX = printer port (0-3)

**Return Registers:**
AX = status
  0001h Jetstream present
  else non-Jetstream port

**See also:** Functions F1h to F5h

### INT 17h Function F1h

*NorthNet Jetstream*

***PRINT DATA BUFFER***

**Registers at call:**
AH = F1h
CX = data buffer length
DX = printer port (0-3)
DS:SI -> data buffer

**Return Registers:**
AX = status
0000h printer not ready (see also Function 02h above)
other printing started

**See also:** Functions F2h, F3h, F5h; Standard Function 00h (above)

### INT 17h Function F2h

*NorthNet Jetstream*

***GET PRINT PROGRESS STATUS***

**Registers at call:**
AH = F2h
DX = printer port (0-3)

**Return Registers:**
AX = status
0000h prior print request finished
other number of characters left to print

**See also:** Functions F1h and F3h, standard Function 02h (above)

### INT 17h Function F3h

*NorthNet Jetstream*

***ABORT PRINT OPERATION***

**Registers at call:**
AH = F3h
DX = printer port (0-3)

**Return Registers:**
AX = number of unprinted characters due to abort

**See also:** Functions F1h and F4h

### INT 17h Function F4h

*NorthNet Jetstream*

***SET COMPLETION (POST) ADDRESS***

**Registers at call:**
AH = F4h
DX = printer port (0-3)
DS:DS -> FAR post address (called with interrupts on)

**Return Registers:**
none

**See also:** Functions F1h and F3h

### INT 17h Function F5h

*NorthNet Jetstream*

***PRINT DATA BUFFER FROM EXTENDED MEMORY***

**Registers at call:**
AH = F5h
CX = data buffer length
DX = printer port (0-3)
DS:SI -> data buffer (32-bit physical address)

**Return Registers:**
AX = status
0000h printer not ready (see also Function 02h above)
other printing started

**See also:** Function F1h

## Postscript converters

The PostScript language (a trademark of Adobe Systems Incorporated) traces its history back to 1976 and the pioneering graphics firm of Evans & Sutherland, where it was known as "the Design System" according to the preface written by John Warnock for Adobe's 1985 Language Reference Manual. Warnock, a co-founder of Adobe, was manager of the E&S group that developed the original version.

While the language's name is proprietary to Adobe, the language itself has become a standard for high-quality publication processing, and a number of programs that can accept instructions written in this Forth-like page description language have appeared from other sources. Such programs are the subject of this section.

---

### INT 05h Function 454Ah
PSPS v2.01
(undocumented)

### *EJECT PAGE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 454Ah ('EJ') | none |

**Program:** PSPS is a shareware PostScript PrintScreen utility by A.N.D. Technologies.

**Details:** This function sends a ^D to the current printer.

**Conflicts:** Bound Exceeded (Chapter 3), Print Screen (Chapter 4)

**See also:** Functions 4E57h, 5053h, and 554Eh

---

### INT 05h Function 4E57h
PSPS v2.01
(undocumented)

### *SET PARAMETERS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4E57h ('NW') | buffer filled |
| ES:SI -> 11-byte buffer containing new settings | |

**Conflicts:** Bound Exceeded (Chapter 3), Print Screen (Chapter 4)

**See also:** Functions 454Ah and 5053h, T2PS INT 17h Function 63h

---

### INT 05h Function 5053h
PSPS v2.01
(undocumented)

### *GET PARAMETERS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5053h ('PS') | none |
| ES:SI -> 11-byte buffer for settings | |

**Conflicts:** Bound Exceeded (Chapter 3), Print Screen (Chapter 4)

**See also:** Functions 4E57h and 554Eh, T2PS INT 17h Function 64h

---

### INT 05h Function 554Eh
PSPS v2.01
(undocumented)

### *UNINSTALL*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 554Eh ('UN') | none |

**Program:** PSPS is a shareware PostScript PrintScreen utility by A.N.D. Technologies.

**Conflicts:** Bound Exceeded (Chapter 3), Print Screen (Chapter 4)

**See also:** Functions 454Ah and 4E57h, T2PS INT 17h Function 62h

---

### INT 17h Function 03h
Emulaser ELTSR
(undocumented)

### *INSTALL INTERRUPT HANDLERS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 03h | BX = *unknown* |
| | CX = *unknown* |

**Program:** ELTSR is the resident portion of the Emulaser PostScript emulator by Vertisoft Systems, Inc.

**Conflicts:** PC-MOS/386 (below)

**See also:** Functions 04h and 0Eh, Disk Spool II INT 1Ah Function E5h (below)

---

### INT 17h Function 04h
Emulaser ELTSR
(undocumented)

### *BEGIN CAPTURING OUTPUT*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 04h | none |

**Details:** This function has no effect unless ELTSR is deactivated (see Function 0503h).

**See also:** Functions 03h and 0503h, Disk Spool II INT 1Ah Function E5h (below)

---

### INT 17h Function 0500h
### UNDETERMINED FUNCTION

*Emulaser ELTSR*
*(undocumented)*

**Registers at call:**
AX = 0500h
*others, if any, unknown*

**Return Registers:**
AX = unload status (0001h safe to unload, 0002h not safe)
BX = *unknown*
CX = PSP segment of ELTSR
DX = activity flag (0000h disabled, 0001h capturing, 0002h *printing*)
SI = *unknown*
DI = *unknown*

**See also:** Function 04h, INT 1Ah Function E5h

---

### INT 17h Function 0501h
### UNHOOK INTERRUPTS

*Emulaser ELTSR*
*(undocumented)*

**Registers at call:**
AX = 0501h

**Return Registers:**
(see Function 0500h)

**Details:** This function restores the interrupt vectors without checking whether they have been hooked by later programs; it should only be called if ELTSR reports that it is safe to unload.

**See also:** Functions 04h, 0500h, and 0503h; Disk Spool II INT 1Ah Function E5h (below)

---

### INT 17h Function 0502h
### SET UNKNOWN VALUE

*Emulaser ELTSR*
*(undocumented)*

**Registers at call:**
AX = 0502h
BL = Emulaser port (31h=LPT1, 32h=LPT2, 33h=LPT3)
CL = *unknown*
DL = *unknown*

**Return Registers:**
(see Function 0500h)

**See also:** Functions 04h and 0500h, Disk Spool II INT 1Ah Function E5h (below)

---

### INT 17h Function 0503h
### DEACTIVATE

*Emulaser ELTSR*
*(undocumented)*

**Purpose:** This function appears to place ELTSR in an inactive state.

**Registers at call:**
AX = 0503h

**Return Registers:**
(see Function 0500h)

**See also:** Functions 04h, 0500h, and 0501h; Disk Spool II INT 1Ah Function E5h (below)

---

### INT 17h Function 06h
### UNDETERMINED FUNCTION

*Emulaser ELTSR*
*(undocumented)*

**Registers at call:**
AH = 06h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 05h, Disk Spool II INT 1Ah Function E5h (below)

---

### INT 17h Function 07h
### OPEN CAPTURE FILE

*Emulaser ELTSR*
*(undocumented)*

**Registers at call:**
AH = 07h
ES:DX -> ASCIZ filename to be opened

**Return Registers:**
*unknown*

**Details:** New output will be appended to the specified file.

**See also:** Function 08h, Disk Spool II INT 1Ah Function E5h (below)

### INT 17h Function 08h
### *CLOSE CAPTURE FILE*

*Emulaser ELTSR (undocumented)*

**Purpose:** Close the file previously opened by Function 07h.

**Registers at call:**
AH = 08h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 07h, Disk Spool II INT 1Ah Function E5h (below)

### INT 17h Function 09h
### *PRINT CAPTURE FILE*

*Emulaser ELTSR (undocumented)*

**Registers at call:**
AH = 09h
BX = *unknown*
CX = *unknown*
DX = printer port (01h COM1,
  02h COM2, 05h LPT1, 06h LPT2)

**Return Registers:**
AX = status
  00h successful
  FFh failed

**Program:** ELTSR is the resident portion of the Emulaser PostScript emulator by Vertisoft Systems, Inc.

**Details:** This function calls through to INT 1Ah Function E401h, and thus requires that either ELSPL or Disk Spool II be installed.

**See also:** Function 0Ah, Disk Spool II INT 1Ah Functions E401h and E5h (below)

### INT 17h Function 0Ah
### *SET UNKNOWN FILENAME*

*Emulaser ELTSR (undocumented)*

**Registers at call:**
AH = 0Ah
ES:BX -> *unknown* buffer
CX = length of buffer

**Return Registers:**
*unknown*

**Details:** This function copies the specified name into the buffer passed to ELSPL as the filename by Function 09h.

**See also:** Function 09h, Disk Spool II INT 1Ah Function E5h (below)

### INT 17h Function 0Bh
### *GET UNKNOWN VALUE*

*Emulaser ELTSR (undocumented)*

**Registers at call:**
AH = 0Bh

**Return Registers:**
AX:BX -> *unknown*

**See also:** Function 0Ah

### INT 17h Function 0Ch
### *SET UNKNOWN FLAG*

*Emulaser ELTSR (undocumented)*

**Registers at call:**
AH = 0Ch
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 0Bh

### INT 17h Function 0Dh
### *GET TRUE ScrlLk STATE*

*Emulaser ELTSR (undocumented)*

**Purpose:** Determine the actual state of ScrlLk even when Emulaser is controlling the ScrlLk light as its activity indicator.

**Registers at call:**
AH = 0Dh

**Return Registers:**
AX = state (0000h off, 0010h on)

**See also:** Function 0Ch, BIOS INT 16h Function 02h (Chapter 4)

---

**INT 17h Function 0Eh** *Emulaser ELTSR*

***BACKGROUND PROCESSING*** *(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AH = 0Eh | none |

**Program:** ELTSR is the resident portion of the Emulaser PostScript emulator by Vertisoft Systems, Inc.

**Details:** This function is called by ELTSR on every INT 08h to allow data to be processed in the background, but may also be called by applications to give Emulaser additional CPU time.

**See also:** Function 0Dh

---

**INT 17h Function 62h** *T2PS v1.0*

***UNINSTALL*** *(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AH = 62h | none |

**Program:** T2PS is a shareware ASCII-to-PostScript converter by A.N.D. Technologies.

**See also:** Functions 63h and 64h, PSPS INT 05h Function 554Eh

---

**INT 17h Function 63h** *T2PS v1.0*

***SET PARAMETERS*** *(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AH = 63h | none |
| ES:SI -> new settings (see Table 15-5) | |

**See also:** Functions 62h and 64h, PSPS INT 05h Function 4E57h

**15-5 Format of settings:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | LPT port number (0=LPT1, etc.) |
| 02h | WORD | page heigh in points |
| 04h | WORD | page width in points |
| 06h | WORD | top margin in points |
| 08h | WORD | bottom margin in points |
| 0Ah | WORD | left margin in points |
| 0Ch | WORD | right margin in points |
| 0Eh | WORD | font size in points |
| 10h | WORD | tab size |
| 12h | WORD | timeout in clock ticks |

---

**INT 17h Function 64h** *T2PS v1.0*

***GET PARAMETERS*** *(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AH = 64h | ES:SI buffer filled |
| ES:SI -> buffer for settings (see Table 15-5) | |

**See also:** Functions 62h and 63h, PSPS INT 05h Function 5053h

## Print Spoolers—Disk Spool II

Disk Spool II is a shareware disk-based print spooler by Budget Software Company.

The version 2.07+ and 4.0x APIs are also supported by Vertisoft's Emulaser utility ELSPL, as that is a licensed version of Disk Spool II.

---

**INT 1Ah Function A0h**
***INSTALLATION CHECK***

*Disk Spool II v2.07+*
*(undocumented)*

**Registers at call:**
AH = A0h

**Return Registers:**
AH = B0h if installed
AL = *pending INT 1Ah Function D0h subfunction if nonzero*
ES = code segment
ES:BX -> name of current spool file
ES:SI -> current despool file
CL = despooler state (00h disabled, 41h enabled)
CH = spooler state (00h disabled, 41h enabled)
DL = despooler print state (00h currently active, 41h standing by)
DH = *unknown* (00h or 41h)
DI = *unknown* (0000h or 0001h)

**See also:** Functions ABh, C0h, D0h, and E1h

---

**INT 1Ah Function ABh**
***INSTALLATION CHECK***

*Disk Spool II v1.83*
*(undocumented)*

**Registers at call:**
AH = ABh

**Return Registers:**
AH = BAh if installed
AL = *pending INT 1Ah Function ADh subfunction if nonzero*
ES = code segment
ES:BX -> name of current spool file
ES:SI -> current despool file
CL = despooler state (00h disabled, 41h enabled)
CH = spooler state (00h disabled, 41h enabled)
DL = despooler print state (00h actively printing, 41h standing by)
DH = *unknown* (00h or 41h)
DI = *unknown* (0000h or 0001h)

**See also:** Functions A0h, ACh, ADh, and E1h

---

**INT 1Ah Function ACh**
***INSTALLATION CHECK***

*Disk Spool II v1.83*
*(undocumented)*

**Registers at call:**
AH = ACh

**Return Registers:**
(see Function ABh)

**Details:** This function is identical to Function ABh.
**See also:** Functions A0h, ABh, and ADh

---

**INT 1Ah Function ADh**
***FUNCTION CALLS***

*Disk Spool II v1.83*
*(undocumented)*

**Registers at call:**
AH = ADh
AL = function code
    02h enable spooler only
    03h enable the despooler
    04h disable the despooler
    08h inhibit popup menu
    09h enable popup menu
    0Ah *unknown*

0Bh disable the spooler
0Ch *start despooler after last successfully printed document*
0Dh *start despooler at the exact point where it last left off*
0Eh pop up the menu
0Fh *unknown*
11h *unknown*
14h *unknown*
15h *unknown*
16h *unknown*
17h *unknown*
18h *unknown*
19h *unknown*
20h *clear file pointed to by the despooler*
21h *unknown*
22h *unknown*
23h *unknown*
30h *unknown*

**Return Registers:**
AH = 00h if successful

**See also:** Functions ABh and D0h

---

### INT 1Ah Function C0h
### *ALTERNATE INSTALLATION CHECK*

*Disk Spool II v2.07+ (undocumented)*

**Registers at call:**
AH = C0h

**Return Registers:**
(see Function A0h)

**Details:** This call is identical to Function A0h.

**See also:** Functions A0h, ABh, and D0h

---

### INT 1Ah Function D0h
### *FUNCTION CALLS*

*Disk Spool II v2.07+ (undocumented)*

**Registers at call:**
AH = D0h
AL = function code
01h enable spooler and despooler
02h enable spooler only
03h enable despooler at beginning of file
04h disable the despooler
05h disable the despooler and spooler
06h clear the spool file
08h inhibit the popup menu
09h enable the popup menu
0Ah *unknown* (called by Disk Spool's INT 21h handler)
0Bh disable the spooler
0Ch start despooler after last successfully printed document
0Dh start despooler at the exact point where it last left off
0Eh pop up the menu
0Fh *unknown*
11h *start new spool file* (called by Disk Spool's INT 21h handler when a program terminates)
14h *unknown*
15h delete despool file and reset *unknown flag*
16h *unknown* (writes something to unknown file)
17h *unknown* (writes something to despool file, then reads something

else and *unknown*)
18h *unknown* (reads something from despool file, and then *unknown*)
19h *unknown* (creates/truncates spool file)
20h clear file pointed to by the despooler
21h *unknown* (writes something to unknown file)
22h *unknown* (writes something to spool file if spooler/despooler using same file)
23h *unknown* (opens/creates unknown file, then *unknown*)
30h *unknown*
31h *unknown*
32h beep
33h *append CRLF to spool file*
34h *unknown*
35h *unknown*
36h *unknown*
37h *append CRLF to spool file and start a new spool file*
38h *unknown*
40h *unknown* (v4.05)
41h *unknown* (v4.05)
51h *unknown* (called by Disk Spool's INT 21h handler)
52h *unknown* (called by Disk Spool's INT 21h handler)
57h *unknown*
5Ah *unknown* (v4.05)
5Bh *unknown* (v4.05)
5Ch *unknown* (v4.05)

**Return Registers:**
*unknown*, vary by function

**See also:** Functions A0h and ADh

---

### INT 1Ah Function E0h

*Disk Spool II v4.0x*

### *ENABLE/DISABLE*

**Registers at call:**
AH = E0h
AL = subfunction
- 01h enable spooler
- 02h disable spooler
- 03h enable despooler
- 04h disable despooler

CL = printer port (01h COM1, 02h COM2, 05h LPT1, 06h LPT2)

**Return Registers:**
AH = status
- 00h successful
- F0h printer port not managed by Disk Spool II
- FFH failed

**See also:** Functions A0h, E1h, E301h, and E401h

---

### INT 1Ah Function E1h

*Disk Spool II v4.0x*

### *GET STATUS*

**Registers at call:**
AH = E1h
CL = printer port (01h COM1, 02h COM2, 05h LPT1, 06h LPT2)

**Return Registers:**
AH = status
- 00h successful
  - CL = despooler state (00h disabled, 41h enabled)
  - CH = spooler state (00h disabled, 41h enabled)
  - DL = despooler activity (00h standing by, 41h printing)
  - ES:BX -> ASCIZ name of current spool file (or next if AutoSpool or AutoDespool enabled)
  - ES:SI -> ASCIZ name of current despool file

ES:DI -> 3-byte file extension used by Disk Spool II
F0h printer port not managed by Disk Spool II

**See also:** Functions A0h, E0h, and E2h

---

### INT 1Ah Function E2h

*Disk Spool II v4.0x*

### *GET SPOOL FILES*

**Registers at call:**
AH = E2h
AL = which
  01h first
  02h next (can only call after "first")
CL = printer port (01h COM1, 02h COM2, 05h LPT1, 06h LPT2)

**Return Registers:**
AH = status
  00h successful
    ES:BX -> ASCIZ filename
  F0h no (more) spool files
  FFh failed

**See also:** Functions E0h and E1h

---

### INT 1Ah Function E301h

*Disk Spool II v4.0x*

### *GET SPOOL FILE STATUS*

**Registers at call:**
AX = E301h
ES:BX -> ASCIZ filename (max 32 chars)

**Return Registers:**
AH = status
  00h successful
    ES:SI -> spool file status record (see Table 15-6)
  F0h not a spool file
  FFh failed

**See also:** Functions E0h, E302h, and E401h

**15-6 Format of spool file status record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | hour of creation or last update |
| 01h | BYTE | minute of creation or last update |
| 02h | BYTE | year-1980 of creation or last update |
| 03h | BYTE | month of creation or last update |
| 04h | BYTE | day of creation or last update |
| 05h | BYTE | total number of copies to print |
| 06h | BYTE | number of copies already printed |
| 07h | BYTE | printer port (01h COM1, 02h COM2, 05h LPT1, 06h LPT2) |
| 08h | BYTE | save status (00h delete after printing, 01h save) |
| 09h | BYTE | file status<br>01h done printing, but being saved<br>02h on hold<br>03h queued for printing<br>04h being spooled<br>05h being despooled (i.e. printed) |
| 0Ah | 16 BYTEs | ASCIZ description |
| 1Ah | 2 WORDs | file size in bytes (high,low) |
| 1Eh | 2 WORDs | bytes left to print (high,low) |

---

### INT 1Ah Function E302h

*Disk Spool II v4.0x*

### *UPDATE SPOOL FILE*

**Registers at call:**
AX = E302h
ES:BX -> ASCIZ filename (max 32 chars)
ES:SI -> spool file status record (see Table 15-6)

**Return Registers:**
AH = status
  00h successful

F0h not a spool file
FFh failed

**See also:** Functions E0h, E301h, and E401h

---

### INT 1Ah Function E401h

*Disk Spool II v4.0x*

***SPOOL EXISTING FILE***

**Registers at call:**
AX = E401h
ES:BX -> ASCIZ filename (max 32 chars)
CL = printer port (01h COM1, 02h COM2, 05h LPT1, 06h LPT2)

**Return Registers:**
AH = status
00h successful
FFh failed

**See also:** Functions E1h, E302h, and E402h

---

### INT 1Ah Function E402h

*Disk Spool II v4.0x*
*(undocumented)*

***SPOOL EXISTING FILE***

**Registers at call:**
AX = E402h
ES:BX -> ASCIZ filename (max 32 chars)
CL = printer port (01h COM1, 02h COM2, 05h LPT1, 06h LPT2)

**Return Registers:**
AH = status
00h successful
FFh failed

**See also:** Functions E1h, E302h, and E401h

---

### INT 1Ah Function E5h

*Emulaser ELSPL.COM*
*(undocumented)*

***UNDETERMINED FUNCTION***

**Registers at call:**
AH = E5h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Program:** ELSPL.COM is a licensed version of Disk Spool II which is distributed as part of Vertisoft's Emulaser PostScript emulator.

**See also:** Function A0h, Emulaser INT 17h Function 03h (above)

---

### INT 1Ah Function EEh

*Disk Spool II v4.05*
*(undocumented)*

***UNDETERMINED FUNCTION***

**Registers at call:**
AH = EEh
AL = *printer port*
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function E1h

## Print Spoolers—PCSpool

PCSpool is a print spooler published by *PC* Magazine.

---

### INT 17h Function C0h

*PC Magazine PCSpool*

***GET CONTROL BLOCK ADDRESS***

**Registers at call:**
AH = C0h
DX = printer port (0-3)

**Return Registers:**
ES:BX -> control block (see Table 15-7)

**See also:** Function C1h

**15-7 Format of control block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | printer number |
| 02h | WORD | address of printer status port |
| 04h | WORD | number of first record in queue |
| 06h | WORD | number of last record in queue |
| 08h | DWORD | characters already printed |
| 0Ch | DWORD | number of characters remaining |
| 10h | DWORD | pointer to dequeue buffer |
| 14h | DWORD | previous count of characters printed |
| 18h | DWORD | number of clock ticks taken to print them |
| 1Ch | WORD | offset of next character to output |
| 1Eh | WORD | offset of next character to print |
| 20h | WORD | pointer to spooling queue record |
| 22h | BYTE | current spooling status |
| 23h | BYTE | current printer status (see Table 15-8) |
| 24h | BYTE | current control record type |
| 25h | WORD | observed printer speed |
| 27h | WORD | characters to print per service |
| 29h | BYTE | 01h if disk write needed |
| 2Ah | BYTE | 01h if queued data should be flushed |
| 2Bh | BYTE | 01h to update cps status |

**15-8 Values for printer status:**

| | |
|---|---|
| 00h | OK |
| 01h | not ready |
| 02h | paused with message |
| 03h | paused |
| 04h | initializing |
| FEh | non-existent port |
| FFh | not spooled |

## INT 17h Function C1h

*PC Magazine PCSpool*

### *BUILD PAUSE CONTROL RECORD*

| Registers at call: | Return Registers: |
|---|---|
| AH = C1h<br>DX = printer port (0-3)<br>DS:SI -> ASCIZ string to save for display | none |

**Details:** This function flushes any pending writes.

**See also:** Functions C0h, C2h, and C5h

## INT 17h Function C2h

*PC Magazine PCSpool*

### *FLUSH PENDING WRITES*

| Registers at call: | Return Registers: |
|---|---|
| AH = C2h<br>DX = printer port (0-3) | none |

**See also:** Function C3h

### INT 17h Function C3h
*PC Magazine PCSpool*

### *CANCEL PRINTER QUEUE (FLUSH ALL QUEUED OUTPUT)*

| Registers at call: | Return Registers: |
|---|---|
| AH = C3h<br>DX = printer port (0-3) | none |

**See also:** Functions C2h and C7h

### INT 17h Function C4h
*PC Magazine PCSpool*

### *QUERY SPOOLER ACTIVE (INSTALLATION CHECK)*

| Registers at call: | Return Registers: |
|---|---|
| AH = C4h | DI = B0BFh if installed<br>SI = segment |

### INT 17h Function C5h
*PC Magazine PCSpool*

### *JOB SKIP PRINTER QUEUE*

| Registers at call: | Return Registers: |
|---|---|
| AH = C5h<br>DX = printer port (0-3) | none |

**Details:** This function cancels all print jobs up to the pause record.
**See also:** Function C1h

### INT 17h Function C6h
*PC Magazine PCSpool*

### *CHECK PRINTER QUEUE STATUS*

| Registers at call: | Return Registers: |
|---|---|
| AH = C6h<br>DX = printer port (0-3) | AX = status<br>0000h printer not active or at pause<br>0001h printer busy |

### INT 17h Function C7h
*PC Magazine PCSpool*

### *CLOSE QUEUE*

| Registers at call: | Return Registers: |
|---|---|
| AH = C7h<br>DX = printer port (0-3) | none |

**See also:** Function C3h

## Print Spoolers—PRINT

PRINT is the file-based print spooler included with MS-DOS.

### INT 15h Function 2000h
*DOS 3+ PRINT.COM (undocumented)*

### *DISABLE CRITICAL REGION FLAG*

**Purpose:** Stop setting user flag on entry to PRINT critical region.

| Registers at call: | Return Registers: |
|---|---|
| AX = 2000h | none |

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Function 2001h, INT 2Ah Function 8700h

### INT 15h Function 2001h
*DOS 3+ PRINT.COM (undocumented)*

### *SET CRITICAL REGION FLAG*

**Purpose:** Specify a user flag which PRINT should set to let an interested application know it is in a critical region.
**Registers at call:**
AX = 2001h

| | **Return Registers:** |
|---|---|
| ES:BX -> byte which is to be incremented while in a DOS call | none |

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Function 2000h, INT 2Ah Function 8700h

---

### INT 2Ah Function 8700h
### *BEGIN BACKGROUND PRINTING*

*PRINT*
*Callout (undocumented)*

**Purpose:** This function is used to inform interested programs that PRINT is about to start its background processing, and allow those programs to postpone the processing if necessary.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 8700h | CF clear if OK to print in background now |
| CF clear | CF set if background printing not allowed at this time |

**Details:** When PRINT gains control and want to begin printing, it calls this function. If CF is clear on return, PRINT begins its background processing, and calls Function 8701h when it is done. If CF is set on return, PRINT will relinquish control immediately, and will not call Functions 8701h.

PCVENUS (an early network shell by IBM and CMU) hooks this call to prevent background printing while its own code is active.

**See also:** Function 8701h, Network Critical Section Functions 80h to 82h (NI)

---

### INT 2Ah Function 8701h
### *END BACKGROUND PRINTING*

*PRINT*
*Callout (undocumented)*

**Purpose:** This function is used to inform interested programs that PRINT has completed its background processing.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 8701h | none |

**Details:** This function is called by PRINT after it has performed some background printing; it will not be called if Function 8700h returned with CF set.

**See also:** Function 8700h

---

### INT 2Fh Function 00h
### *UNDETERMINED FUNCTION*

*DOS 2.x only PRINT.COM*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 00h | *unknown* |
| *others, if any, unknown* | |

**Details:** DOS 2.x PRINT.COM does not chain to previous INT 2F handler.

Values in AH other than 00h or 01h cause PRINT to return the number of files in the queue in AH.

**Conflicts:** DOS 3+ PRINT, PSPRINT (below)
**See also:** Function 01h

---

### INT 2Fh Function 0080h
### *GIVE PRINT A TIME SLICE*

*DOS 3.1+ PRINT.COM*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 0080h | after PRINT executes |

**Details:** PRINT returns AL=01h if AH=00h but AL is not 80h on entry.
**Conflicts:** DOS 2.x PRINT
**See also:** Function 0104h

---

### INT 2Fh Function 01h

*DOS 2.x only PRINT.COM (undocumented)*

#### *UNDETERMINED FUNCTION*

**Registers at call:**
AH = 01h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** DOS 2.x PRINT.COM does not chain to the previous INT 2Fh handler.

Values in AH other than 00h or 01h cause PRINT to return the number of files in the queue in AH.

**Conflicts:** DOS 3+ PRINT

**See also:** Function 00h

---

### INT 2Fh Function 0100h

*DOS 3+ PRINT.COM*

#### *INSTALLATION CHECK*

**Registers at call:**
AX = 0100h

**Return Registers:**
AL = status
- 00h not installed
- 01h not installed, but not OK to install
- FFh installed

**Conflicts:** DOS 2.x PRINT Function 01h

**See also:** Function 0101h

---

### INT 2Fh Function 0101h

*DOS 3+ PRINT.COM*

#### *SUBMIT FILE FOR PRINTING*

**Registers at call:**
AX = 0101h
DS:DX -> submit packet (see Table 15-9)

**Return Registers:**
CF clear if successful
  AL = status
    01h added to queue
    9Eh now printing
CF set on error
  AX = error code (see Tables 15-10 and 24-2)

**Conflicts:** DOS 2.x PRINT Function 01h

**See also:** Function 0102h

**15-9 Format of submit packet:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | level (must be 00h) |
| 01h | DWORD | pointer to ASCIZ filename (no wildcards) |

**15-10 PRINT error codes:**

| | |
|---|---|
| 01h | invalid function |
| 02h | file not found |
| 03h | path not found |
| 04h | out of file handles |
| 05h | access denied |
| 08h | print queue full |
| 09h | spooler busy |
| 0Ch | name too long |
| 0Fh | invalid drive |

---

### INT 2Fh Function 0102h

*DOS 3+ PRINT.COM*

#### *REMOVE FILE FROM PRINT QUEUE*

**Registers at call:**
AX = 0102h

DS:DX -> ASCIZ filename
(wildcards allowed)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 15-10)

**Conflicts:** DOS 2.x PRINT Function 01h
**See also:** Functions 0101h and 0103h

---

### INT 2Fh Function 0103h
*DOS 3+ PRINT.COM*
### *CANCEL ALL FILES IN PRINT QUEUE*

**Registers at call:**
AX = 0103h

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 15-10)

**Conflicts:** DOS 2.x PRINT Function 01h
**See also:** Function 0102h

---

### INT 2Fh Function 0104h
*DOS 3+ PRINT.COM*
### *FREEZE PRINT QUEUE TO READ JOB STATUS*

**Purpose:** Get the list of print jobs, temporarily suspending PRINT's activities to avoid changing the list while it is being examined.

**Registers at call:**
AX = 0104h

**Return Registers:**
CF clear if successful
DX = error count since status last read
DS:SI -> print queue
CF set on error
AX = error code (see Table 15-10)

**Details:** The print queue is an array of 64-byte ASCIZ fiienames terminated by an empty filename; the first name is the file currently being printed.

Printing is stopped until Function 0105h is called to prevent the queue from changing while the filenames are being read.

**Conflicts:** DOS 2.x PRINT Function 01h
**See also:** Functions 0101h and 0105h

---

### INT 2Fh Function 0105h
*DOS 3+ PRINT.COM*
### *RESTART PRINT QUEUE AFTER STATUS READ*

**Purpose:** Restart PRINT's activities once an application finishes examining the print queue.

**Registers at call:**
AX = 0105h

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 15-10)

**Conflicts:** DOS 2.x PRINT Function 01h
**See also:** Function 0104h

---

### INT 2Fh Function 0106h
*DOS 3.3+ PRINT.COM*
### *GET PRINTER DEVICE*

**Purpose:** Determine which device, if any, PRINT is currently using for output.

**Registers at call:**
AX = 0106h

**Return Registers:**
CF set if files in print queue
AX = error code 0008h (queue full)
DS:SI -> device driver header
CF clear if print queue empty
AX = 0000h

**Details:** This function has been documented for DOS 5+, but was not documented for prior versions.

This function can be used to allow a program to avoid printing to the printer on which PRINT is currently performing output.

**Conflicts:** DOS 2.x PRINT Function 01h

**See also:** Function 0104h

## Printer Redirectors

A number of utilities are available to redirect output which has been sent to a printer into a file instead.

---

### INT 17h
*LPTx v5.x*

***INSTALLATION CHECK***

**Registers at call:**
DX = 0ABCh

**Return Registers:**
AX = AAAAh
DX = BAAAh
ES = code segment of resident portion

**Conflicts:** BIOS INT 17h (above)

---

### INT 17h
*LPTx v6.x*

***INSTALLATION CHECK***

**Registers at call:**
DX = 0B90h

**Return Registers:**
DX = ABBBh
ES = code segment of resident portion

**Conflicts:** BIOS INT 17h (above)

---

### INT 17h
*LPTx v7.x*

***INSTALLATION CHECK***

**Registers at call:**
DX = 0B91h

**Return Registers:**
DX = ABCBh
ES = code segment of resident portion

**Conflicts:** BIOS INT 17h (above)

---

### INT 17h
*LPTx v4.x*

***INSTALLATION CHECK***

**Registers at call:**
DX = 0F5Fh

**Return Registers:**
AX = AAAAh
DX = F555h
ES = code segment of resident portion

**Conflicts:** BIOS INT 17h (above)

---

### INT 7Ah
*PRINDIR*
*Obsolete*

***API (moved to INT 7Ch)***

**Conflicts:** GO32 (Chapter 37), AutoCAD Device Interface (Chapter 66), Canon Image Scanner (Chapter 66), IBM 3270 Workstation Program (NI), Novell NetWare (NI), Topware (NI), X.PC (NI)

**See also:** PRINDIR INT 7Ch

---

### INT 7Ch
*PRINDIR v8.02+*

***API***

**Registers at call:**
AH = function
    00h set output to Invisible mode

- 03h set output to LPT3
- 04h set output to COM1
- 05h set output to COM2
- 06h set output to COM3
- 07h set output to COM4
- 08h set output to screen
- 09h set output to file
- 0Ah change file path
    - ES:DI -> new path
    - CX = length of path (max 41, but not range-checked)
- 0Bh change file name
    - ES:DI -> new filename
    - CX = length of new name (max 41, but not range-checked)
- 0Ch set redirected device
    - (01h = LPT1 ... 03h = LPT3, 04h = COM1 ... 07h = COM4)
- 0Dh force buffer dump
- 0Eh set display color
    - AL = new color attribute (see INT 10h Function 08h)
- 0Fh set popup hotkey
    - AL = hotkey scancode
- 10h toggle byte display
    - AL = new state (00h off, 01h on)
- 11h change buffer flush delay
    - AL = timer ticks before flush

**Return Registers:**

AX = status
- 0000h successful
- 0001h invalid function or nothing in buffer to be dumped (documented as "nonzero = error")

**Program:** PRINDIR is a shareware printer-redirection program by J.M. Allen Creations.

**Details:** The installation check consists of testing for the ASCIZ signature "PRINDIR N.NN" (where N.NN is the version number) two bytes beyond the start of the interrupt handler.

Prior versions of PRINDIR used INT 7Ah instead of INT 7Ch.

**Conflicts:** GO32 (Chapter 37), IBM REXX88PC (Chapter 66), SK-UPPS (NI)

**See also:** PRINDIR INT 7Ah

## Miscellaneous

### INT 17h Function 03h
*PC-MOS/386 v5.01*

### *PRINT SPOOLER - PRINT STRING*

**Purpose:** Send an entire string of characters to the print spooler with a single call.

**Registers at call:**

AH = 03h
DX = printer port number
CX = number of characters to print
DS:SI -> string to be printed

**Return Registers:**

AH = printer status (see Table 15-1)
CX = number of characters actually printed

**Program:** PC-MOS/386 v5.01 is a multitasking, multiuser MS-DOS 5.0-compatible operating system by The Software Link, Inc.

**Conflicts:** Emulaser ELTSR (above)

**See also:** Function FFh, Standard Functions 00h to 02h (above)

---

### INT 17h Function 2000h

*PC Paint Plus 2.0*

### *PRINTER DRIVER - INSTALLATION CHECK*

**Registers at call:**
AX = 2000h
BL = *printer number*

**Return Registers:**
AX = status (see Table 15-11)
BX = driver version number (BH=major,BL=minor)
CH = *unknown* (00h)
CL = *unknown*
DX = *unknown* (0100h)

**Details:** This function also enables the remaining 200xh functions.
**See also:** Functions 2001h to 2006h

**15-11 Values for PC Paint status:**

| | |
|---|---|
| 0000h | successful |
| 0001h | *invalid printer* |
| 0002h | *unknown* |
| 0003h | invalid subfunction |
| 0005h | driver disabled, must call function 00h first |
| 0009h | unknown printer error |
| 000Bh | printer not selected |
| 000Ch | printer out of paper |
| 000Eh | error while writing to serial printer |
| 000Fh | *unknown* |
| 0010h | invalid function number |
| 0011h | value out of range |

---

### INT 17h Function 2001h

*PC Paint Plus 2.0*

### *PRINTER DRIVER - SET UNKNOWN FLAG*

**Registers at call:**
AX = 2001h
BL = *printer number*

**Return Registers:**
AX = status (see Table 15-11)

**See also:** Function 2000h

---

### INT 17h Function 2002h

*PC Paint Plus 2.0*

### *PRINTER DRIVER - GET INFORMATION*

**Registers at call:**
AX = 2002h
BL = *printer number*
CL = desired information
    00h printer type
    01h paper size
        DX = size index
    02h *unknown*
    03h *printer information*
        DX = *unknown*
        ES:BX -> buffer for *unknown information* (min 134 bytes)

**Return Registers:**
AX = status (see Table 15-11)
ES:DI -> ASCIZ printer name
ES:DI -> ASCIZ paper size description
BX = *unknown*

**See also:** Function 2000h

---

### INT 17h Function 2003h

*PC Paint Plus 2.0*

***PRINTER DRIVER - UNDETERMINED FUNCTION***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 2003h | AX = status (see Table 15-11) |
| ES:BX -> *unknown* | |

---

### INT 17h Function 2004h

*PC Paint Plus 2.0*

***PRINTER DRIVER - GET UNKNOWN INFORMATION***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 2004h | AX = status (see Table 15-11) |
| BL = *printer number* | ES:DI -> *unknown* |

**See also:** Function 2000h

---

### INT 17h Function 2005h

*PC Paint Plus 2.0*

***PRINTER DRIVER - ADVANCE PRINTER TO NEXT PAGE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 2005h | AX = status (see Table 15-11) |
| BL = *printer number* | |

**Details:** This function also clears the flag set by function 2001h.

**See also:** Function 2006h

---

### INT 17h Function 2006h

*PC Paint Plus 2.0*

***PRINTER DRIVER - ADVANCE PRINTER TO NEXT PAGE AND SHUT DOWN***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 2006h | AX = status (see Table 15-11) |
| BL = *printer number* | |

**Details:** This function also clears flag set by function 01h and disables functions other than 00h.

**See also:** Function 2005h

---

### INT 17h Function 2007h

*PC Paint Plus 2.0*

***PRINTER DRIVER - NOT IMPLEMENTED***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 2007h | AX unchanged |
| BL = *printer number* | |

**See also:** Function 2000h

---

### INT 2Fh Function 00h

*PSPRINT*
*(undocumented)*

***PRINT JOB CONTROL***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 00h | *unknown* |
| *others, if any, unknown* | |

**Conflicts:** MS-DOS PRINT (above)

**See also:** Function 2000h

---

### INT 2Fh Function CD00h

*LaserPort Interface*

***INSTALLATION CHECK***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = CD00h | AL = status<br>FFh installed |

**Conflicts:** SWELL (Chapter 49), Intel Image Processing Interface (Chapter 55); see also Table 2-2

**See also:** LaserPort Function CD0Fh

**INT 2Fh Function CD0Fh** *LaserPort Interface*

***EXECUTE FUNCTION***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = CD0Fh | AL = status |
| BL = function | 00h success |
| 01h enable | |
| 02h disable | |
| 03h *unknown* | |

**Conflicts:** SWELL (Chapter 49), Intel Image Processing Interface (Chapter 55); see also Table 2-2

**See also:** LaserPort Function CD00h

## CHAPTER 16

# *Disk Compression*

As storage needs have skyrocketed, a class of programs intended to ease their users' space problems has appeared. These programs all operate to compress files automatically when writing them to the disk, and expand the files back when they are accessed, thus making it possible to store more information on an existing disk drive. Most of them claim to achieve an average of 2:1 compression, allowing storage of 80 megabytes of data on a 40-megabyte drive, but in practice few achieve this figure over the entire range of data that is stored. In particular, files which have already been compressed with an efficient file compressor program cannot gain significant additional reduction in size, and may even become larger unless the disk compressor detects that possibility and makes no attempt to compress such files.

This chapter covers the most significant disk compression programs available at the time of writing, including the new Microsoft RealTime Compression Interface introduced with MS-DOS version 6.0.

## DIET

DIET is a transparent file compressor/decompressor by Teddy Matsumoto.

---

### INT 21h Function 37D0h — *DIET v1.43e*

***TSR INSTALLATION CHECK***

**Registers at call:**
AX = 37D0h
BX = 899Dh ('DI' + 'ET')

**Return Registers:**
AL = FFh if not present as TSR (default return valuefrom DOS)
AX = 0000h if installed as a TSR
CX = 899Dh ('DI' + 'ET')
DX = version ID

**See also:** Functions 37D1h, 37D2h, 37D4h, 37D6h, 37DFh, 4BF0h, and 4BF1h

---

### INT 21h Function 37D1h — *DIET v1.43e*

***GET DIET.EXE RESIDENT SEGMENT***

**Registers at call:**
AX = 37D1h
BX = 899Dh ('DI' + 'ET')

**Return Registers:**
AX = 0000h
CX = code segment of TSR part of DIET.EXE
DX = memory block segment of TSR DIET.EXE (0000h if installed as device driver)

**See also:** Functions 37D0h and 37DFh

### INT 21h Function 37D2h

*DIET v1.43e*

### *GET TSR CONTROL FLAGS*

**Registers at call:**
AX = 37D2h
BX = 899Dh ('DI' + 'ET')

**Return Registers:**
AX = 0000h
DL = control flag (00h active, else disabled)
DH = skip flag (nonzero while TSR active)

**See also:** Functions 37D0h, 37D3h, and 37D4h

### INT 21h Function 37D3h

*DIET v1.43e*

### *SET TSR CONTROL FLAGS*

**Registers at call:**
AX = 37D3h
BX = 899Dh ('DI' + 'ET')
DL = control flag (00h active, else disabled)
DH = skip flag (00h)

**Return Registers:**
AX = 0000h

**See also:** Functions 37D0h, 37D2h, and 37D5h

### INT 21h Function 37D4h

*DIET v1.43e*

### *GET TSR OPTIONS*

**Registers at call:**
AX = 37D4h
BX = 899Dh ('DI' + 'ET')

**Return Registers:**
AX = 0000h
DX = TSR options (see Table 16-1)

**See also:** Functions 37D0h, 37D2h, and 37D5h

**16-1 Bitfields for TSR options:**

| | |
|---|---|
| bit 0 | automated compression of DIETed file |
| bit 1 | automated compression of newly-created file |
| bit 2 | suppress DIET message |
| bit 3 | display original file size |
| bits 4-15 | reserved (0) |

### INT 21h Function 37D5h

*DIET v1.43e*

### *SET TSR OPTIONS*

**Registers at call:**
AX = 37D5h
BX = 899Dh ('DI' + 'ET')
DX = TSR options (see Table 16-1)

**Return Registers:**
AX = 0000h

**See also:** Functions 37D0h, 37D3h, and 37D4h

### INT 21h Function 37D6h

*DIET v1.43e*

### *GET TEMPORARY DIRECTORY NAMES*

**Registers at call:**
AX = 37D6h
BX = 899Dh ('DI' + 'ET')

**Return Registers:**
AX = 0000h
DS:DX -> name of temporary directory or
0000h:0000h for current directory

**See also:** Functions 37D0h and 37D7h

### INT 21h Function 37D7h

*DIET v1.43e*

### *SET TEMPORARY DIRECTORY NAMES*

**Registers at call:**
AX = 37D7h
BX = 899Dh ('DI' + 'ET')
DS:DX -> ASCIZ name of temporary

directory (max 61 chars)
0000h:0000h for current directory

**Return Registers:**
AX = 0000h

**Details:** The specified directory name must include a drive letter and end with a backslash.
**See also:** Functions 37D0h and 37D6h

---

## INT 21h Function 37DCh

*DIET v1.43e*

### *SET ADDRESS OF EXTERNAL PROCEDURE*

**Registers at call:**
AX = 37DCh
BX = 899Dh ('DI' + 'ET')
DS:DX -> external procedure (see Table 16-2)

**Return Registers:**
AX = 0000h

**Details:** The resident code will call the specified external procedure at the beginning of decompression and when compression is exited on failure.
**See also:** Function 37DDh

### 16-2 External procedure calling convention:

**Registers at call:**
STACK: WORD class
FFFDh creation failed for unknown reasons
FFFEh creation failed due to lack of space
FFFFh file creation error
else file handle of DIETed file to be decompressed
DWORD pointer to compressed filename
DWORD pointer to decompressed or temporary filename

**Return Registers:**
SI,DI,BP,DS,ES must be preserved by external procedure

---

## INT 21h Function 37DDh

*DIET v1.43e*

### *RELEASE EXTERNAL PROCEDURE*

**Registers at call:**
AX = 37DDh
BX = 899Dh ('DI' + 'ET')

**Return Registers:**
none

**Details:** This function unlinks the external procedure specified by Function 37DCh.
**See also:** Function 37DCh

---

## INT 21h Function 37DEh

*DIET v1.43e*

### *READ EMS STATUS*

**Registers at call:**
AX = 37DEh
BX = 899Dh ('DI' + 'ET')

**Return Registers:**
AX = 0000h
CX = EMS status
0000h not used
0001h used as work area
0002h used for code and as work area
DX = EMM handle when EMS is in use

---

## INT 21h Function 37DFh

*DIET v1.43e*

### *UNLOAD TSR*

**Registers at call:**
AX = 37DFh
BX = 899Dh ('DI' + 'ET')

**Return Registers:**
AX = status
0000h successful
00FFh failed

**See also:** Function 37D0h

### INT 21h Function 4BF0h
*DIET v1.10+ (Overlay Mode)*
### *INSTALLATION CHECK*

**Registers at call:**
AX = 4BF0h

**Return Registers:**
CF clear if installed
AX = 899Dh ('DI' + 'ET')

**See also:** Functions 37D0h and 4BF1h

### INT 21h Function 4BF1h
*DIET v1.10+ (Overlay Mode)*
### *EXPAND PROGRAM*

**Registers at call:**
AX = 4BF1h

**Return Registers:**
*unknown*

**See also:** Functions 37D0h and 4BF0h

## DoubleSpace

DBLSPACE.BIN is the resident driver for DoubleSpace, the disk-compression software bundled with MS-DOS 6.0.

### INT 21h Function 4404h
*DBLSPACE.BIN*
### *IOCTL - FLUSH OR INVALIDATE INTERNAL CACHES*

**Registers at call:**
AX = 4404h
BL = drive number (00h = default, 01h = A:, etc)
CX = 000Ah (size of DSPACKET structure)
DS:DX -> DSPACKET structure (see Table 16-3)

**Return Registers:**
CF clear if IOCTL successful—check DSPACKET for actual status
AX = number of bytes actually transferred
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Conflicts:** Stacker (below), DoubleDisk (below), xDISK (Chapter 10), COMBI-disk (Chapter 17)
**See also:** Function 4405h, INT 2Fh Function 4A11h Subfunction 0000h

**16-3 Format of DSPACKET structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | signature 444Dh ("DM(") |
| 02h | BYTE | command code<br>46h ('F') flush internal caches<br>49h ('I') flush and invalidate internal caches |
| 03h | WORD | result code<br>(return) 4F4Bh ("OK(") if successful, else unchanged |
| 05h | 5 BYTEs | padding |

### INT 21h Function 4405h
*DBLSPACE.BIN (undocumented)*
### *IOCTL - FLUSH OR INVALIDATE INTERNAL CACHES*

**Registers at call:**
AX = 4405h
BL = drive number (00h = default, 01h = A:, etc)
CX = 000Ah (size of DSPACKET structure)
DS:DX -> DSPACKET structure (see Table 16-3)

**Return Registers:**
CF clear if IOCTL successful—check DSPACKET for actual status
AX = number of bytes actually transferred
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Details:** This call is identical to the documented Function 4404h.
**Conflicts:** xDISK (Chapter 10), ST-01 SCSI.SYS (Chapter 11), COMBI-disk (Chapter 17)
**See also:** Function 4404h, INT 2Fh Function 4A11h Subfunction 0000h

### INT 2Fh Function 4A11h, Subfunction 0000h

*DBLSPACE.BIN*

***"GetVersion(" - INSTALLATION CHECK***

**Registers at call:**
AX = 4A11h
BX = 0000h

**Return Registers:**
AX = 0000h (successful)
BX = 444Dh ("DM(")
CL = first drive letter used by DBLSPACE (0=A:)
CH = number of drive letters used by DBLSPACE
DX = internal DBLSPACE.BIN version number bit 15 set if DBLSPACE.BIN has not yet been relocated to final position in memory (i.e. DBLSPACE.SYS /MOVE)

**See also:** Function 4A11h Subfunctions 0001h, 0002h, 0003h, 0005h, 0007h, and FFFFh; INT 21h Function 4404h

### INT 2Fh Function 4A11h, Subfunction 0001h

*DBLSPACE.BIN*

***"GetDriveMapping(" - GET DRIVE MAPPING***

**Registers at call:**
AX = 4A11h
BX = 0001h
DL = drive number (0=A:)

**Return Registers:**
AX = status (see also Table 16-4)
0000h successful
BL = host drive (bit 7 set if specified drive is compressed)
BH = DoubleSpace sequence number
other error code (0101h) (see Table 16-4)

**Details:** The compressed volume file for the specified compressed drive is **host:\DBLSPACE.sequence**.

**See also:** Function 4A11h Subfunctions 0000h and 0002h

**16-4 Values for status:**

| | |
|---|---|
| 0000h | successful |
| 0100h | bad function |
| 0101h | invalid drive |
| 0102h | not a compressed drive |
| 0103h | drive already swapped |
| 0104h | drive not swapped |

### INT 2Fh Function 4A11h, Subfunction 0002h

*DBLSPACE.BIN*

***"Swap Drive(" - SWAP DRIVE LETTERS OF CVF AND HOST DRIVE***

**Registers at call:**
AX = 4A11h
BX = 0002h
DL = drive number (0=A:) of compressed drive to swap with its host

**Return Registers:**
AX = status (0000h,0101h,0102h,0103h) (see Table 16-4)

**Details:** This function is intended for use by DBLSPACE.EXE only.

**See also:** Function 4A11h Subfunction 0000h

### INT 2Fh Function 4A11h, Subfunction 0003h

*DBLSPACE.BIN*
*(undocumented)*

***GET DEVICE DRIVER ENTRY POINTS***

**Registers at call:**
AX = 4A11h
BX = 0003h
CL = drive number (0=A:) of compressed drive

**Return Registers:**
CL = FFh on error (not compressed drive)
= *other host drive*
ES:SI -> device driver's strategy routine
ES:DI -> device driver's interrupt routine
BX destroyed

**Details:** The official documentation states that this function is reserved for use by SMARTDRV; this function and Function 4A11h Subfunction 0004h allow SMARTDRV to apply the same device driver wrapper to DoubleSpaced drives that it applies to regular block devices.
**See also:** Function 4A11h Subfunctions 0000h and 0004h, SMARTDRV Function 4A10h Subfunction 0007h (Chapter 17)

---

### INT 2Fh Function 4A11h, Subfunction 0004h
### *SET DEVICE DRIVER ENTRY POINTS*

*DBLSPACE.BIN*
*(undocumented)*

**Registers at call:**
AX = 4A11h
BX = 0004h
CL = drive number (0=A:) of compressed drive
ES:SI -> device driver strategy routine to call for drive
ES:DI -> device driver interrupt routine to call for drive
DX = *host for compressed drive*

**Return Registers:**
CL = FFh on error
BX destroyed

**Details:** The official documentation states that this function is reserved for use by SMARTDRV; this function and Function 4A11h Subfunction 0003h allow SMARTDRV to apply the same device driver wrapper to DoubleSpaced drives that it applies to regular block devices.
**See also:** Function 4A11h Subfunctions 0000h and 0003h, SMARTDRV Function 4A10h Subfunction 0007h (Chapter 17)

---

### INT 2Fh Function 4A11h, Subfunction 0005h
### *"ActivateDrive(" - MOUNT COMPRESSED DRIVE*

*DBLSPACE.BIN*

**Registers at call:**
AX = 4A11h
BX = 0005h
DL = drive number (0=A:) to assign to new drive
ES:SI -> activation record (see Table 16-5)

**Return Registers:**
status returned in activation record (see Table 16-5)

**See also:** Function 4A11h Subfunctions 0000h and 0006h

**16-5 Format of activation record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | signature "MD(" (4Dh 44h) |
| 02h | BYTE | 4Dh ('M') mount command |
| 03h | BYTE | error code (set to FFh before calling)<br>01h drive letter not available for DoubleSpace<br>02h drive letter already in use<br>03h no more disk units (increase MaxRemovableDrives in .INI)<br>09h CVF too fragmented |
| 04h | BYTE | host drive number (0=A:) |
| 05h | *unknown* | DISK_UNIT structure (not documented) |

---

### INT 2Fh Function 4A11h, Subfunction 0006h
### *"DeactivateDrive(" - UNMOUNT COMPRESSED DRIVE*

*DBLSPACE.BIN*

**Registers at call:**
AX = 4A11h
BX = 0006h
DL = drive number (0=A:) tounmount

**Return Registers:**
AX = status (0000h,0102h) (see Table 16-4)

**See also:** Function 4A11h Subfunctions 0000h and 0005h

### INT 2Fh Function 4A11h, Subfunction 0007h *DBLSPACE.BIN*

### *"GetDriveSpace(" - GET SPACE AVAILABLE ON COMPRESSED DRIVE*

**Registers at call:**
AX = 4A11h
BX = 0007h
DL = compressed drive number (0=A:)

**Return Registers:**
AX = status (0000h,0102h) (see also Table 16-4)
0000h successful
DS:SI -> free space record (see Table 16-6)

**See also:** Function 4A11h Subfunctions 0000h and 0008h

**16-6 Format of free space record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | total number of sectors in drive's sector heap |
| 04h | DWORD | number of free sectors in drive's sector heap |

### INT 2Fh Function 4A11h, Subfunction 0008h *DBLSPACE.BIN*

### *"GetFileFragmentSpace(" - GET SIZE OF FRAGMENT HEAP*

**Registers at call:**
AX = 4A11h
BX = 0008h
DL = compressed drive number (0=A:)

**Return Registers:**
AX = status (0000h,0102h) (see also Table 16-4)
0000h successful
BX = maximum entries in File Fragment heap
CX = available entries in File Fragment heap

**See also:** Function 4A11h Subfunctions 0000h, 0007h, and 0009h

### INT 2Fh Function 4A11h, Subfunction 0009h *DBLSPACE.BIN*

### *"GetExtraInfo(" - DETERMINE NUMBER OF DISK_UNIT STRUCTURES*

**Registers at call:**
AX = 4A11h
BX = 0009h
DL = compressed drive number (0=A:)

**Return Registers:**
AX = status (see Table 16-4)
0000h successful
CL = number of DISK_UNIT structures allocated (see Function 4A11h Subfunction 0005h)

**See also:** Function 4A11h Subfunctions 0000h and 0008h

### INT 2Fh Function 4A11h, Subfunction FFFEh *DBLSPACE.BIN (undocumented)*

### *RELOCATE*

**Registers at call:**
AX = 4A11h
BX = FFFEh
ES = segment to which to relocate DBLSPACE.BIN

**Return Registers:**
*unknown*

**Details:** This function is called by DBLSPACE.SYS to relocate DBLSPACE.BIN to its final position in memory.

This function also unhooks and discards the code providing both this function and Function 4A11h Subfunction FFFFh.

**See also:** Function 4A11h Subfunction FFFFh

### INT 2Fh Function 4A11h, Subfunction FFFFh *DBLSPACE.BIN (undocumented)*

### *GET RELOCATION SIZE*

**Registers at call:**
AX = 4A11h
BX = FFFFh

**Return Registers:**
AX = number of paragraphs needed by DBLSPACE.BIN

**Details:** This function is used by DBLSPACE.SYS to relocate the DBLSPACE driver to its final position in memory.

**See also:** Function 4A11h Subfunctions 0000h and FFFEh

### INT 2Fh Function 4A13h
### *GET UNKNOWN ENTRY POINTS*

*DBLSPACE.BIN*
*(undocumented)*

**Registers at call:**
AX = 4A13h

**Return Registers:**
AX = 134Ah if supported
ES:BX -> entry point record (see Table 16-7)

**See also:** Function 4A11h Subfunction 0000h

**16-7 Format of entry point record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to FAR function for *unknown* |
| 04h | 5 BYTEs | FAR JUMP instruction to *unknown* |

## Microsoft Real-Time Compression Interface

The Microsoft Real-Time Compression Interface (MRCI, pronounced "merci") is a standard introduced with MS-DOS Version 6.0 to provide a uniform technique for compressing data in real time. MRCI is the compression technique used by DoubleSpace and also by the MSBackup utility of MS-DOS 6.0; a number of third-party utilities are also expected to offer MRCI compatibility.

The MRCI standard defines both ROM-based servers and RAM-based servers (using different interrupt conventions for each) and specifies an API for use by client processes.

### INT 1Ah Function B001h
### *ROM-BASED SERVER*

*MRCI*

**Registers at call:**
AX = B001h
CX = 4D52h ("MR")
DX = 4349h ("CI")

**Return Registers:**
CX = 4943h ("IC") if installed
DX = 524Dh ("RM") if installed
ES:DI -> MRCINFO structure (see Table 16-8)

**Details:** This call is functionally identical to INT 2Fh Function 4A12h, which should be called first, as this call is used for the first, ROM-based MRCI server, while the other call is used for RAM-based servers which may be partially or entirely replacing a prior server.

**See also:** INT 2Fh Function 4A12h

**16-8 Format of MRCINFO structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h 4 | BYTEs | vendor signature<br>"MSFT" Microsoft |
| 04h | WORD | server version (high=major) |
| 06h | WORD | MRCI specification version |
| 08h | DWORD | address of server entry point (see Table 16-10) |
| 0Ch | WORD | bit flags: server capabilities (see Table 16-9) |
| 0Eh | WORD | bit flags: hardware assisted capabilities (see Table 16-9) |
| 10h | WORD | maximum block size supported by server (at least 8192 bytes) |

**16-9 Bitfields for capabilities:**

| | |
|---|---|
| bit 0 | standard compress |
| bit 1 | standard decompress |
| bit 2 | update compress |
| bit 3 | MaxCompress (removed from initial release) |
| bit 4 | reserved |
| bit 5 | incremental decompress |
| bits 6-14 | reserved |
| bit 15 | this structure is in ROM and can't be modified (server capabilities only) |

**16-10 MRCI entry point calling convention:**

**Registers at call:**
DS:SI -> MRCREQUEST structure (see Table 16-11)
CX = type of client (0000h application, 0001h file system)
AX = operation
    0001h perform standard compression
    0002h perform standard decompression
    0004h perform update compression
    0008h perform MaxCompress (not supported in original releases)
    0020h perform incremental decompression
AX = FFFFh to clear flags
    BX = bitmask of flags to clear (set bits in BX are flags to clear)

**Return Registers:**
AX = status
    0000h successful
    0001h invalid function
    0002h server busy, try again
    0003h destination buffer too small
    0004h incompressible data
    0005h bad compressed data format

**Details:** The MRCI driver may chain to a previous driver.

**16-11 Format of MRCREQUEST structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to source buffer |
| 04h | WORD | size of source buffer (0000h = 64K) |
| 06h | WORD | (UpdateCompress only)<br>(call) offset in source buffer of beginning of changed data<br>(return) offset in destination buffer of beginning of changed compressed data |
| 08h | DWORD | pointer to destination buffer<br>must contain original compressed data for UpdateCompress |
| 0Ch | WORD | size of destination buffer (0000h = 64K)<br>any compression: size of buffer for compressed data<br>standard decompression: number of bytes to be decompressed<br>incremental decompression: number of byte to decompress now<br>(return) actual size of resulting data |
| 0Eh | WORD | client compressed data storage allocation size |
| 10h | DWORD | incremental decompression state data<br>set to 00000000h before first incremental decompression call |

**Notes:** The source and destination buffers may not overlap; their sizes should normally be the same.

The application should not update the contents of the MRCREQUEST structure between incremental decompression calls.

### INT 2Fh Function 4A12h

*MRCI*

***RAM-BASED SERVER***

**Registers at call:**
AX = 4A12h
CX = 4D52h ("MR")
DX = 4349h ("CI")

**Return Registers:**
CX = 4943h ("IC") if installed
DX = 524Dh ("RM") if installed
    ES:DI -> MRCINFO structure (see Table 16-8)

**Details:** This call is functionally identical to INT 1Ah Function B001h, but should be called first, as the latter call is used for the first, ROM-based MRCI server, while this call is used for RAM-based servers which may be partially or entirely replacing a prior server.

**See also:** INT 1Ah Function B001h

## NewSpace

NewSpace is a TSR by Isogon Corporation which automatically compresses all files as they are written and decompresses them as they are read. NewSpace was written in 1988 or 1989, and thus uses some function numbers which have since been claimed by MS-DOS.

---

### INT 21h Function 4410h
### *ENABLE DRIVER*

*NewSpace*
*(undocumented)*

**Registers at call:**
AX = 4410h
BX = FFFFh

**Return Registers:**
none

**Details:** Compressed files are not accessible unless the driver is enabled.
**Conflicts:** MS-DOS (Chapter 24)
**See also:** Function 4411h

---

### INT 21h Function 4411h
### *DISABLE DRIVER*

*NewSpace*
*(undocumented)*

**Registers at call:**
AX = 4411h
BX = FFFFh

**Return Registers:**
none

**Details:** Compressed files are not accessible unless the driver is enabled.
**Conflicts:** MS-DOS (Chapter 24)
**See also:** Function 4410h

---

### INT 21h Function 4412h
### *INSTALLATION CHECK*

*NewSpace*
*(undocumented)*

**Registers at call:**
AX = 4412h
BX = FFFFh

**Return Registers:**
AX = PSP segment of NewRes (resident driver for NewSpace)
BX:DX -> *unknown*
CX = *unknown*

**Conflicts:** DR-DOS (Chapter 26)
**See also:** Function 4411h

---

### INT 21h Function 4413h
### *GET UNKNOWN VALUE*

*NewSpace*
*(undocumented)*

**Registers at call:**
AX = 4413h
BX = FFFFh

**Return Registers:**
AX = code segment of NewRes (resident driver for NewSpace)
BX = offset of *unknown*

**Conflicts:** DR-DOS (Chapter 26)
**See also:** Function 4412h

---

### INT 21h Function 4414h
### *DEBUGGING DUMP*

*NewSpace*
*(undocumented)*

**Registers at call:**
AX = 4414h
BX = FFFFh

**Return Registers:**
debugging dump written to X:\NEWSPACE.SMP

**Conflicts:** DR-DOS (Chapter 26)
**See also:** Functions 4413h and 44FFh

---

### INT 21h Function 44FFh
### *UNDETERMINED FUNCTION*

*NewSpace*
*(undocumented)*

**Registers at call:**
AX = 44FFh
BX = FFFFh
DX = *unknown*

**Return Registers:**
none

**See also:** Function 4414h

## Stacker

Stacker is a disk compression package dreated by Stac Electronics of Carlsbad, CA. The current version for MS-DOS is 3.1; versions also exist for OS/2 and for the MacIntosh. The MS-DOS version is compatible with all versions of MS-DOS and Microsoft Windows and takes advantage of special features present in MS-DOS version 6.0.

Stacker uses a proprietary LZS algorithm which maintains a sliding window of the most recently processed 2048 bytes of data from a file to accomplish its compression. While not the first disk compression program, it has become the de facto standard against which other disk compression packages are judged.

---

### INT 21h Function 4404h

*Stacker*

***GET DEVICE DRIVER ADDRESS AND SET VOLUME NUMBER***

**Registers at call:**
AX = 4404h
BL = drive number (00h = default, 01h = A:, etc.)
CX = 0004h
DS:DX -> DWORD buffer to receive device driver address

**Return Registers:**
CF clear if successful
AX = number of bytes actually transferred
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Details:** In addition to returning the address of the Stacker device driver, this call also sets the volume number at offset 3Eh in the device driver (see INT 25h Function CDCDh).

**Conflicts:** DoubleSpace (above), DoubleDisk (below), xDISK (Chapter 10), COMBI-disk (Chapter 17)

**See also:** INT 25h Function CDCDh

---

### INT 21h Function 4404h

*Stacker*

***GET STACVOL FILE SECTORS***

**Purpose:** Read data from the physical file containing the compressed data for a Stacker logical drive.

**Registers at call:**
AX = 4404h
BL = drive number (00h is current drive)
CX = byte count (i.e., 200h = 1 sector)
DS:DX -> buffer (see Table 16-12)

**Return Registers:**
Data Buffer contains the number of sectors requested from the STACVOL physical file for the drive specified.

**Conflicts:** DoubleSpace (above), DoubleDisk (below), xDISK (Chapter 10), COMBI-disk (Chapter 17)

**16-12 Format of stacker buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | 01CDh |
| 02h | WORD | sector count |
| 04h | DWORD | number of starting sector |
| 08h | DWORD | far pointer to Data Buffer |

---

### INT 25h Function CDCDh

*Stacker*

***GET DEVICE DRIVER ADDRESS***

**Registers at call:**
AX = CDCDh
DS:BX -> buffer for address (see Table 16-13)
CX = 0001h
DX = 0000h

**Return Registers:**
AX = CDCDh if Stacker installed
DS:BX buffer filled

**Conflicts:** MS-DOS INT 25h (Chapter 24)

### 16-13 Format of driver address buffer:

| Offset | Size | Description |
|---|---|---|
| 00h | WORD | signature CDCDh |
| 02h | WORD | *unknown* (0001h) |
| 04h | DWORD | pointer to start of Stacker device driver |

### 16-14 Format of device driver:

| Offset | Size | Description |
|---|---|---|
| 00h | WORD | signature A55Ah |
| 02h | WORD | Stacker version * 64h |
| 04h | WORD | offset of volume-specific information offset table<br>(list of WORDs, one per drive, containing offsets to various information) |
| 06h | 56 BYTEs | n/a |
| 3Eh | BYTE | volume number, set after INT 21h Function 4404h<br>(use to index into volume-specific info offset table) |
| 3Fh | 19 BYTEs | n/a |
| 52h | 4 BYTEs | ASCII string "SWAP(" |
| 56h | 26 BYTEs | drive mapping table (one byte for each drive A: through Z:) |

### 16-15 Format of Stacker boot record:

| Offset | Size | Description |
|---|---|---|
| 1F0h | 8 BYTEs | Stacker signature (first byte is CDh) |
| 1F8h | DWORD | pointer to start of Stacker device driver |
| 1FCh | WORD | Stacker volume number |
| 1FEh | WORD | *unknown* |

## SuperStor

SuperStor is a disk-compression program by Addstor.

---

### INT 2Fh Function 1001h
### *SET UNKNOWN VALUE*

*SuperStor PRO SSTORDRV.SYS*
*(undocumented)*

**Registers at call:**
AX = 1001h
BX = *unknown*
DX = *unknown*

**Return Registers:**
none

**See also:** Function F800h

---

### INT 2Fh Function F800h
### *INSTALLATION CHECK*

*SuperStor PRO 2XON.COM*
*(undocumented)*

**Registers at call:**
AX = F800h
CX = 4455h ("DU")
DL = 45h ("E")

**Return Registers:**
AL = FFh if installed
ES:BX -> ASCII signature "Universal Data Exchange"

**Details:** Returns AX=0001h if AL is not 00h or 01h.
**Conflicts:** none, but see Table 2-2
**See also:** Functions 1001h and F801h

---

### INT 2Fh Function F801h
### *UNINSTALL*

*SuperStor PRO 2XON.COM*
*(undocumented)*

**Registers at call:**
AX = F801h
CX = 4455h ("DU")

DL = 45h ("E")
ES:BX = return address if successful

**Return Registers:**
at specified address if successfully removed from memory else
AL = error code
*others, if any, unknown*

**Details:** 2XON returns AX=0001h if AL is not 00h or 01h.
**Conflicts:** none, but see Table 2-2
**See also:** Functions 1001h and F800h

## Miscellaneous

### INT 21h Function 4404h

*DUBLDISK.SYS v2.6*

***GET INFO***

**Registers at call:**
AX = 4404h
BL = drive number of DoubleDisk drive (00h = default, 01h = A:, etc.)
CX = number of bytes (000Ah-0014h, call ignored otherwise)
DS:DX -> data record (see Table 16-16)

**Return Registers:**
CF clear if successful
AX = number of bytes read
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Program:** DUBLDISK.SYS is the device driver portion of DoubleDisk, a disk expander by Vertisoft Systems, Inc.

**Details:** The installation check consists of scanning memory for the signature "FAT 2.6 byte:(", which is immediately followed by a data table.

**Conflicts:** DoubleSpace (above), Stacker (above), xDISK (Chapter 10), COMBI-disk (Chapter 17)

**See also:** MS-DOS Function 440Dh (Chapter 24)

**16-16 Format of data record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | (call) signature 4444h |
| 02h | BYTE | (call) function<br>00h *unknown*<br>01h *unknown* |
| **---function 00h---** | | |
| 02h | BYTE | (return) *unknown* |
| 03h | BYTE | (return) *unknown* |
| **---function 01h---** | | |
| 02h | WORD | (return) 4444h |
| 04h | WORD | *allocation unit size* |
| 06h | WORD | *unknown* |
| 08h | WORD | *unknown* |
| 0Ah | BYTE | *unknown* |

**16-17 Format of signature data table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 5 BYTEs | *unknown* |
| 05h | BYTE | first drive number |
| 06h | BYTE | number of drives |
| 07h | | *unknown* |

### INT 21h Function 69FFh, Subfunction 0000h

*CUBIT v4.00*
*(undocumented)*

### *GET CUBIT INT 21h HANDLER*

**Registers at call:**
AX = 69FFh
DX = 0000h
BX = CB00h (magic value)

**Return Registers:**
ES:BX -> CUBITR.EXE handler for INT 21h

**Details:** The installation check consists of testing that the first eight bytes at the returned interrupt handler are EBh 07h "CUBITR(" (a short jump around the signature followed by the signature); the byte following the signature (i.e. ES:[BX+8]) indicates whether CUBITR is active (01h) or disabled (00h).

**See also:** Function 69FFh Subfunction CFBFh

### INT 21h Function 69FFh, Subfunction CFBFh

*CUBIT v4.00*
*(undocumented)*

### *UNINSTALL*

**Registers at call:**
AX = 69FFh
DX = CFBFh
CX = EFCFh
BX = CB00h (magic value)

**Return Registers:**
ES:BX -> CUBITR.EXE handler for INT 21h
CX = status
  2020h successful
  2222h failed

**Details:** If DX is neither 0000h nor CFBFh on entry, some other code is executed.

**See also:** Function 69FFh Subfunction 0000h

### INT 21h Function DCh

*PCMag PCMANAGE/DCOMPRES*

### *TURN ON/OFF*

**Registers at call:**
AH = DCh
DX = new state (0000h on, 0001h off)

**Return Registers:**
none

**Conflicts:** Novell NetWare (NI)

**See also:** Function FEDCh

### INT 21h Function FEDCh

*PCMag PCMANAGE/DCOMPRES*

### *INSTALLATION CHECK*

**Registers at call:**
AX = FEDCh

**Return Registers:**
AX = CDEFh if installed

**Program:** The PCMANAGE/DCOMPRES combination from *PC* Magazine permits infrequently-used files to be compressed to save space and be transparently expanded when accessed.

**Conflicts:** MS-DOS OEM Function (Chapter 24), DoubleDOS (Chapter 49), Viruses (Chapter 59)

**See also:** Function DCh, dLite INT 2Dh Function 10h

### INT 2Dh Function 10h

*dLite 1.0+*

### *GET PARAMETER BLOCK ADDRESS*

**Registers at call:**
AL = 10h
AH = AMIS multiplex number for dLite (see Chapter 50)

**Return Registers:**
CF clear if successful
  ES:BX -> parameter block (see Table 16-18)
CF set on error

**Program:** dLite is a shareware TSR by Rainer Schuetze which transparently expands compressed files when they are read.

**See also:** dLite Functions 11h and 12h, PCMANAGE INT 21h Function FEDCh

**16-18 Format of parameter block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | TSR flags<br>bit 0: deny FCB access |

| | | |
|---|---|---|
| | | bit 1: dLite sleeping rather than activated<br>bit 2: always indicate original filesize when reading directory entries, rather than only for specified programs<br>bits 3-7 reserved |
| 01h | WORD | maximum number of programs needing original filesize |
| 03h | WORD | current number of programs needing original filesize |
| 05h | WORD | maximum number of files that can be handled by dLite (should be the same as FILES= in CONFIG.SYS) |
| 07h | WORD | offset (in the same segment as the parameter block) of the table of programs needing the original filesize (8 bytes each, without path or extension, uppercase, and zero terminated if shorter than 8 bytes) |

---

## INT 2Dh Function 11h

*dLite 1.0+*

### *CHECK FOR dPressed FILE AND GET ORIGINAL SIZE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AL = 11h | CF clear if successful |
| AH = AMIS multiplex number for dLite (see Chapter 50) | DX:AX = size of uncompressed file |
| BX = file handle | CF set on error (not dPressed file) |

**See also:** dLite Functions 10h and 12h

---

## INT 2Dh Function 12h

*dLite 1.0+*

### *CHECK FOR dPressed FILE AND GET COMPRESSED SIZE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AL = 12h | CF clear if successful |
| AH = AMIS multiplex number for dLite (see Chapter 50) | DX:AX = size of compressed file |
| BX = file handle | CF set on error (not dPressed file) |

**See also:** dLite Functions 10h and 11h

## CHAPTER 17

# *Disk Caches*

The purpose of disk cache software is to reduce the time required to access data that is stored on disk by keeping the data most recently used in RAM. Since studies have indicated that such data is most likely to be accessed again soon, this eliminates disk access time when the data is again needed. MS-DOS includes one disk cache program, SMARTDRV, and many third-party suppliers offer them. This chapter covers the interrupts used by such programs.

## FAST!

FAST! is a disk cache by Future Computing Systems and marketed by BLOC Publishing Corp.

### INT 13h Function 80h *FAST! v4.02+*

***API***

**Registers at call:**

AH = 80h
CX = 6572h
DX = 1970h
ES:BX -> request packet (see Table 17-1)
AL = function number (see Table 17-3)

**Return Registers:**

AH = status (except function 06h)
    00h successful
    01h invalid function
    05h not supported by the installed variant
CF clear if successful
CF set on error
AL may be destroyed

**See also:** Functions 8001h, 8006h, and 8007h

**17-1 Format of request packet:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to 19-byte signature string (see Table 17-2) |
| 04h | DWORD | pointer to buffer for data (if needed by function) |

**17-2 Version 4.04-5.03 signature string:**

13h 07h 06h 08h 11h 18h 0Fh 0Eh 02h 18h 13h 08h 0Bh 08h 01h 00h 04h 08h 15h

**17-3 FAST! function numbers:**

| | |
|---|---|
| 01h | get cache information (see separate entry) |
| 04h | disable cache |
| 05h | enable cache and reset statistics |
| 06h | installation check (see separate entry) |
| 07h | unhook interrupts (see separate entry) |
| 09h | flush cache |
| 0Ah | (v4.02+) enable staged writes |
| 0Bh | (v4.02+) disable staged writes |
| 0Ch | (v4.02+) enable beep on flush |

| | |
|---|---|
| 0Dh | (v4.02+) disable beep on flush |
| 0Eh | *unknown* |
| 0Fh | *unknown* |
| 10h | (v4.12+) enable hotkeys |
| 11h | (v4.12+) disable hotkeys |
| 12h | (v4.13+) set idle delay |
| 13h | (v4.13+) set flush dirty percentage |
| 14h | (v5.00+) enable mouse checks |
| 15h | (v5.00+) disable mouse checks |
| 16h | (v5.00d+) reduce cache size to minimum |
| 17h | (v5.00d+) increase cache size to maximum |

---

## INT 13h Function 8001h

*FAST! v4.02+*

### *GET CACHE INFORMATION*

**Registers at call:**
AX = 8001h
CX = 6572h
DX = 1970h
ES:BX -> request packet (see Table 17-4)

**Return Registers:**
AH = 00h if successful

**See also:** Functions 80h and 8006h

**17-4 Format of request packet:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | -> 19-byte signature string (see table 17-2) |
| 04h | DWORD | -> buffer for cache information (see Table 17-5) |

**17-5 Format of cache information (v5.00-5.03):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | binary version number of FAST! (v5.00 = 01F4h) |
| 02h | BYTE | revision letter (61h = X.XXa, 62h = X.XXb, etc.) |
| 03h | BYTE | FAST! variant<br>(01h = FASTE, 02h = FASTX BIOS, 04h = FASTC, 20h = FASTX XMS) |
| 04h | DWORD | total number of read requests |
| 08h | DWORD | number of physical disk reads |
| 0Ch | DWORD | grabbed hash buckets |
| 10h | DWORD | "st_386mem" |
| 14h | DWORD | total number of writes (only counted when staging enabled) |
| 18h | DWORD | number of physical disk writes (only when staging enabled) |
| 1Ch | DWORD | number of write errors while flushing cache |
| 20h | WORD | flags<br>bit 0: beep on flush<br>bit 3: hotkeys enabled<br>bit 4: mouse idle check enabled<br>bit 8: caching enabled<br>bit 13: *unknown* |
| 22h | WORD | flags<br>bit 0: *unknown*<br>bit 1: staged writes enabled |
| 24h | WORD | *unknown* |
| 26h | WORD | maximum cache size in KB |
| 28h | WORD | minimum cache size in KB |
| 2Ah | WORD | segment of first cache buffer (FASTC) |

| | | |
|---|---|---|
| | | segment of EMS page frame (FASTE)<br>XMS handle (FASTX XMS) |
| 2Ch | WORD | number of hash buckets containing no entries |
| 2Eh | WORD | number of hash buckets containing one entry |
| 30h | WORD | number of hash buckets containing two entries |
| 32h | WORD | number of hash buckets containing three entries |
| 34h | WORD | number of hash buckets containing four entries |
| 36h | WORD | number of hash buckets containing five entries |
| 38h | WORD | maximum contiguous sectors |
| 3Ah | WORD | hash factor |
| 3Ch | WORD | number of paragraphs of memory used below 1M |
| 3Eh | WORD | entries per hash bucket |
| 40h | WORD | idle delay in seconds |
| 42h | 2 BYTEs | *unknown* |
| 44h | WORD | staged write threshold percentage |
| 46h | 2 BYTEs | *unknown* |
| 48h | WORD | number of dirty sectors |
| 4Ah | WORD | number of staged write buffers |
| 4Ch | WORD | current cache size in KB |
| 4Eh | WORD | beep frequency in Hz |
| 50h | WORD | *unknown* |
| 52h | WORD | *unknown* |

---

### INT 13h Function 8006h

*FAST! v4.02+*

### *INSTALLATION CHECK*

**Registers at call:**
AX = 8006h
CX = 6572h
DX = 1970h
ES:BX -> request packet (see Table 17-6)

**Return Registers:**
AX = 1965h if installed

**See also:** Functions 80h, 8001h, and 8007h

**17-6 Format of request packet:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | -> 19-byte signature string (see Table 17-2) |

---

### INT 13h Function 8007h

*FAST! v4.02+*

### *UNHOOK INTERRUPTS*

**Registers at call:**
AX = 8007h
CX = 6572h
DX = 1970h
ES:BX -> request packet (see Table 17-6)

**Return Registers:**
AX = 1965h if installed

**See also:** Functions 80h and 8006h

## HyperDisk

HyperDisk is a shareware disk cache by HyperWare (Roger Cross).

---

### INT 13h Function 8EEDh

*HyperDisk v4.01+*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 8EEDh
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Functions 8EEEh and 8EEFh, INT 2Fh Function DF00h, SWBIOS Function EEh (Chapter 10)

---

## INT 13h Function 8EEEh
*HyperDisk v4.01+*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 8EEEh

**Return Registers:**
CF set
AX = CS of HyperDisk resident code
*others, if any, unknown*

**Details:** This function is identical to Function 8EEFh in HYPERDKX v4.21-4.30.
**See also:** Functions 8EEDh and 8EEFh, SWBIOS Function EEh (Chapter 10)

---

## INT 13h Function 8EEFh
*HyperDisk v4.01+*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 8EEFh

**Return Registers:**
CF set
AX = CS of HyperDisk resident code
*others, if any, unknown*

**Details:** This function is identical to Function 8EEEh in HYPERDKX v4.21-4.30.
**See also:** Functions 8EEDh and 8EEEh, SWBIOS Function EEh (Chapter 10)

---

## INT 2Fh Function DF00h
*HyperWare programs*

### *INSTALLATION CHECK*

**Registers at call:**
AX = DF00h
BX = product code
    4248h ('BH') HyperStb
    4448h ('DH') HyperDisk v4.20+
    4B48h ('KH') HyperKey
    5348h ('SH') HyperScreen
CX = 0000h
DX = 0000h

**Return Registers:**
AL = status
    00h not installed
    FFh multiplex number in use
        CX = 5948h ('YH') if selected product installed
        **---HyperDisk---**
        BX = code segment of resident portion
        DX = HyperDisk local data version

**Details:** AH=DFh is the default; if it is already in use by some other program, HyperWare programs then scan multiplex numbers from C0h through FFh.
**Conflicts:** none, but see Table 2-2
**See also:** Functions DF01h and DF02h, INT 13h Function 8EEDh

---

## INT 2Fh Function DF01h
*HyperDisk v4.50+*

### *GET CURRENT CACHE STATE*

**Registers at call:**
AX = DF01h
BX = 4448h ('DH')

**Return Registers:**
AX = 0000h if function supported
    BX = number of cache buffers in use
    CX = number of cache buffers which have been modified
    DL = caching flags (see Table 17-7)

**Conflicts:** none, but see Table 2-2
**See also:** Functions DF00h and DF02h

**17-7 Bitfields for caching flags:**

| | |
|---|---|
| bit 0 | staged writes enabled for floppy disks |
| bit 1 | staged writes enabled for hard disks |
| bit 2 | writes verified on floppy disks |
| bit 3 | writes verified on hard disks |

| | |
|---|---|
| bit 4 | reserved (0) |
| bit 5 | reserved (0) |
| bit 6 | floppy caching enabled |
| bit 7 | all caching functions enabled |

---

### INT 2Fh Function DF02h
*HyperDisk v4.50+*

### *SET CACHE STATE*

**Registers at call:**
AX = DF02h
BX = 4448h ('DH')
DL = new caching flags (see Table 17-7)

**Return Registers:**
AX = 0000h if supported
BX = number of cache buffers in use
CX = number of cache buffers which have been modified
DL = previous caching flags (see Table 17-7)

**Details:** AH=DFh is the default; if it is already in use by some other program, HyperWare programs then scan multiplex numbers from C0h through FFh.
**Conflicts:** none, but see Table 2-2
**See also:** Functions DF00h and DF01h

## NOW!

NOW! is a disk cache by Vertisoft Systems, Inc.

---

### INT 13h Function 7B00h
*NOW! v3.05*

### *GET INFORMATION*

**Registers at call:**
AX = 7B00h
CX:DX -> 1F8h-byte buffer for information record (see Table 17-8)
**See also:** Functions 7B02h and EFh

**Return Registers:**
AX = 0000h
BX = segment of main resident code
ES = *unknown*

**17-8 Format of information record:**

| *Offset* | *Size* | *Description* | | |
|---|---|---|---|---|
| 00h | 80 BYTEs | name of directory from which NOW! was started | | |
| 50h | 424 BYTEs | *unknown* | | |
| 81h | ? BYTEs | array of bytes for *unknown* | | |
| F7h | 250 BYTEs | array of 25 entries, *one per drive* | | |
| | | Offset | Size | Description |
| | | 00h | 2 BYTEs | *unknown* |
| | | 02h | WORD | *unknown* |
| | | 04h | WORD | *unknown* |
| | | 06h | 4 BYTEs | *unknown* |
| 1F1h7 | BYTEs | *unknown* | | |

---

### INT 13h Function 7B01h
*NOW! v3.05*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 7B01h
**See also:** Function 7B00h

**Return Registers:**
DX = segment of *unknown*

---

### INT 13h Function 7B02h
*NOW! v3.05*

### *SET INFORMATION*

**Registers at call:**
AX = 7B02h

BX = segment of *unknown* (10h above a PSP)
CX:DX -> 1F8h-byte information record (see Table 17-8)

**Return Registers:**
*unknown*

**Details:** NOW! grabs the INT 24h value from the PSP reached via the segment in BX.
**See also:** Function 7B00h

---

*NOW! v3.05*

### INT 13h Function 7B03h
### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 7B03h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Functions 7B00h and 7B04h

---

*NOW! v3.05*

### INT 13h Function 7B04h
### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 7B04h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 7B03h

---

*NOW! v3.05*

### INT 13h Function 7B05h
### *GET ACCESS COUNTS*

**Registers at call:**
AX = 7B05h

**Return Registers:**
BX:AX = *number of physical accesses*
DX:CX = *total disk accesses*

**See also:** Functions 7B00h and 7B06h

---

*NOW! v3.05*

### INT 13h Function 7B06h
### *GET UNKNOWN VALUE*

**Registers at call:**
AX = 7B06h
BX = *unknown*

**Return Registers:**
AX = 0000h
BX = *unknown*

**See also:** Functions 7B05h and 7B07h

---

*NOW! v3.05*

### INT 13h Function 7B07h
### *GET UNKNOWN VALUE*

**Registers at call:**
AX = 7B07h

**Return Registers:**
AX = *unknown*
BX = *unknown*
CX = *unknown*
DX = *unknown*

**See also:** Function 7B06h

---

*NOW! v3.05*

### INT 13h Function 7B08h
### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 7B08h
CX = *unknown* (default 00h)

**Return Registers:**
*unknown*

**See also:** Function 7B00h

---

*NOW! v3.05*

### INT 13h Function EFh
### *UNDETERMINED FUNCTION*

**Registers at call:**
AH = EFh
CX = *unknown*

DL = *unknown*
*others, if any, unknown*
**See also:** Function 7B00h

**Return Registers:**
*unknown*

## PC-Cache

PC-Cache is disk cache included with Central Point Software's PC Tools. Versions 5.x were licensed versions of Super PC-Kwik, and are covered in the Super PC-Kwik section. Version 8.0 also adds the SmartDrive API (see below) for compatibility, while retaining the API described here.

### INT 16h Function FFA5h, Subfunction 1111h

*PC-Cache v6+*

***INSTALLATION CHECK***

**Registers at call:**
AX = FFA5h
CX = 1111h

**Return Registers:**
CH = 00h if installed
  ES:DI -> internal data (see Table 17-9)
  CL = cache state
    01h enabled
    02h disabled

**Conflicts:** OPTIMA Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** QuickCache II INT 13h Function 27h (below), Super PC-Kwik INT 13h Function A0h (below), Super PC-Kwik INT 21h Function 2Bh Subfunction 4358h (below)

**17-9 Format of internal data:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -1Ch | 20 BYTEs | cached drive list, one byte per drive A: to T:<br>each byte is either blank (20h) or drive letter (41h-54h) |
| -8 | BYTE | *unknown* |
| -7 | WORD | number of physical transfers (scaled down to 0000h-7FFFh) |
| -5 | WORD | number of saved transfers (scaled down to 0000h-7FFFh) |
| -3 | 3 BYTEs | *unknown* |

### INT 16h Function FFA5h, Subfunction AAAAh

*PC-Cache v6+*

***ENABLE DELAYED WRITES***

**Registers at call:**
AX = FFA5h
CX = AAAAh

**Return Registers:**
AX = *unknown* (apparently either 0000h or sectors_in_cache - 5)

**Conflicts:** OPTIMA Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** Function FFA5h Subfunction CCCCh

### INT 16h Function FFA5h, Subfunction CCCCh

*PC-Cache v6+*

***FLUSH CACHE AND DISABLE DELAYED WRITES***

**Registers at call:**
AX = FFA5h
CX = CCCCh

**Return Registers:**
AX = *unknown* (apparently either 0000h or sectors_in_cache - 5)

**Details:** Delayed writes are automatically disabled on EXECing (see INT 21h Function 4Bh in Chapter 24) a program named either WIN.CO? or DV.E??; however, delayed writes are not automatically reenabled upon the program's termination in version 6.
**Conflicts:** OPTIMA Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** Function FFA5h Subfunctions AAAAh and FFFFh

### INT 16h Function FFA5h, Subfunction DDDDh

*PC-Cache v6+*

***FLUSH AND DISABLE CACHE***

**Registers at call:**
AX = FFA5h
CX = DDDDh

**Return Registers:**
none

**Conflicts:** OPTIMA Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** Function FFA5h Subfunctions EEEEh and FFFFh

---

### INT 16h Function FFA5h, Subfunction EEEEh

*PC-Cache v6+*

***ENABLE CACHE***

**Registers at call:**
AX = FFA5h
CX = EEEEh

**Return Registers:**
none

**Conflicts:** OPTIMA Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** Function FFA5h Subfunction DDDDh

---

### INT 16h Function FFA5h, Subfunction FFFFh

*PC-Cache v6+*

***FLUSH CACHE***

**Registers at call:**
AX = FFA5h
CX = FFFFh

**Return Registers:**
none

**Conflicts:** OPTIMA Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** Function FFA5h Subfunctions CCCCh and DDDDh, Super PC-Kwik INT 13h Function A1h (below)

## QuickCache II

QUICKCACHE II is a shareware disk cache by P.R. Glassel and Associates, Inc.

---

### INT 13h Function 20h

*QUICKCACHE II v4.20*
*(partially doc.)*

***DISMOUNT***

**Purpose:** Flush any dirty buffers for the specified drive(s) and then discard those sector buffers.

**Registers at call:**
AH = 20h
AL = drive (00h = A:, etc.
or 7Fh for *all removable drives*,
or FFh for all drives)

**Return Registers:**
AX = status (0000h successful)

**Conflicts:** Western Digital Super BIOS (Chapter 10)
**See also:** Functions 21h, 22h, and 28h

---

### INT 13h Function 21h

*QUICKCACHE II v4.20*
*(partially doc.)*

***FLUSH CACHE***

**Purpose:** Immediately write all dirty sectors back to disk.

**Registers at call:**
AH = 21h

**Return Registers:**
AX = status (0000h successful)

**See also:** Functions 25h, 2Eh, and 2Fh

---

### INT 13h Function 22h

*QUICKCACHE II v4.20*
*(partially doc.)*

***ENABLE/DISABLE CACHE***

**Purpose:** Enable/disable caching of all drives.

**Registers at call:**
AH = 22h
AL = new state (00h disabled, 01h enabled)

**Return Registers:**
AX = status (0000h successful)

**See also:** Functions 2Ch, 2Dh, 32h, and 33h; Super PC-Kwik Functions A3h and A4h (below)

---

### INT 13h Function 23h

*QUICKCACHE II v4.20*
*(undocumented)*

***GET UNKNOWN VALUE ADDRESS***

**Registers at call:**
AH = 23h

**Return Registers:**
AX = status (0000h successful)
ES = segment of *unknown* data

### INT 13h Function 24h
*QUICKCACHE II v4.20*
***SET SECTORS***
*(partially doc.)*

**Registers at call:**
AH = 24h
BX = new number of sector buffers in cache

**Return Registers:**
AX = status
0000h successful
0001h failed--size adjusted
8000h cache cannot be resized while enabled

**See also:** Function 36h

### INT 13h Function 25h
*QUICKCACHE II v4.20*
***SET FLUSH INTERVAL***
*(partially doc.)*

**Purpose:** Specify how often the cache should write dirty buffers to disk when buffered writes are enabled.

**Registers at call:**
AH = 25h
BX = interval

**Return Registers:**
AX = status (0000h successful)

**See also:** Functions 21h, 2Ch, and 2Eh

### INT 13h Function 26h
*QUICKCACHE II v4.20*
***UNINSTALL***
*(undocumented)*

**Registers at call:**
AH = 26h

**Return Registers:**
AX = status
0000h successful
0001h-00FFh interrupt vector which was hooked by another TSR

**See also:** Function 27h

### INT 13h Function 27h
*QUICKCACHE II v4.20*
***INSTALLATION CHECK***
*(partially doc.)*

**Registers at call:**
AH = 27h
BX = 0000h

**Return Registers:**
AX = 0000h if installed
BX nonzero if installed
BH = major version
BL = binary minor version

**See also:** Function 26h, PC-Cache INT 16h Function FFA5h Subfunction 1111h (above), Super PC-Kwik Function A0h (below)

### INT 13h Function 28h
*QUICKCACHE II v4.20*
***SET AUTOMATIC DISMOUNT***
*(undocumented)*

**Registers at call:**
AH = 28h
AL = new state (00h disabled, 01h enabled)

**Return Registers:**
AX = status (0000h successful)

**See also:** Function 20h

### INT 13h Function 29h
*QUICKCACHE II v4.20*
***NOP***
*(undocumented)*

**Registers at call:**
AH = 29h

**Return Registers:**
AX = 0000h

### INT 13h Function 2Ah
*QUICKCACHE II v4.20*
***SET BUFFER SIZE***
*(partially doc.)*

**Purpose:** Specify the number of cache sector buffers to dedicate to buffered read and write operations.

**Registers at call:**
AH = 2Ah
AL = buffer size (1-30)

**Return Registers:**
AX = status (0000h successful)

**See also:** Functions 2Ch, 2Dh, 39h, and 3Ah

---

### INT 13h Function 2Bh
### *DRIVE ACCESS SOUNDS*

*QUICKCACHE II v4.20*
*(undocumented)*

**Registers at call:**
AH = 2Bh
AL = new state (00h disabled, 01h enabled)

**Return Registers:**
AX = status (0000h successful)

---

### INT 13h Function 2Ch
### *SET BUFFERED WRITES*

*QUICKCACHE II v4.20*
*(partially doc.)*

**Purpose:** Specify whether the cache should delay disk writes.

**Registers at call:**
AH = 2Ch
AL = new state (00h disabled, 01h enabled)

**Return Registers:**
AX = status (0000h successful)

**Details:** This function enables or disables delayed writes for all drives; use Function 38h to change a single drive.

**See also:** Functions 25h, 2Dh, 2Eh, and 38h

---

### INT 13h Function 2Dh
### *SET BUFFERED READ*

*QUICKCACHE II v4.20*
*(partially doc.)*

**Purpose:** Specify whether the cache should attempt to read ahead of actual requests.

**Registers at call:**
AH = 2Dh
AL = new state (00h disabled, 01h enabled)

**Return Registers:**
AX = status (0000h succesful)

**Details:** This function enables or disables read-ahead for all drives; use Function 37h to change a single drive.

**See also:** Functions 2Ch and 37h

---

### INT 13h Function 2Eh
### *SET FLUSH COUNT*

*QUICKCACHE II v4.20*
*(partially doc.)*

**Purpose:** Specify how many dirty sectors the cache should write after each flush interval (see Function 25h) when buffered writes are enabled.

**Registers at call:**
AH = 2Eh
BX = flush count

**Return Registers:**
AX = status (0000h successful)

**See also:** Functions 21h, 25h, and 2Ch

---

### INT 13h Function 2Fh
### *FORCE IMMEDIATE INCREMENTAL FLUSH*

*QUICKCACHE II v4.20*

**Purpose:** Immediately flush up to "flushcount" dirty sectors to disk as if the flush interval had expired.

**Registers at call:**
AH = 2Fh

**Return Registers:**
AX = status (0000h successful)

**See also:** Functions 21h and 2Eh

---

### INT 13h Function 30h
### *GET INFO*

*QUICKCACHE II v4.20*
*(partially doc.)*

**Registers at call:**
AH = 30h
AL = what to get
    00h system info (see Table 17-10)

01h drive info (see Table 17-11)
02h access frequency (array of 30 words)
03h drive index (array of 32 bytes indicating BIOS drive for DOS drive)
DS:DX -> buffer for info

**Return Registers:**
AX = status (0000h successful, 8000h invalid info specifier)

**17-10 Format of system info:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | flag: cache enabled |
| 01h | BYTE | flag: buffered writes enabled |
| 02h | BYTE | flag: buffered reads enabled |
| 03h | BYTE | flag: sounds enabled |
| 04h | BYTE | flag: autodismount enabled |
| 05h | BYTE | *unknown* |
| 06h | BYTE | flag: *unknown* |
| 07h | BYTE | flag: *unknown* |
| 08h | BYTE | flag: "em_assigned" |
| 09h | BYTE | flag: emulated EMS |
| 0Ah | BYTE | single sector bonus |
| 0Bh | BYTE | "sticky_max" |
| 0Ch | BYTE | write sector bonus |
| 0Dh | BYTE | bonus threshold |
| 0Eh | WORD | flush interval |
| 10h | WORD | flush count |
| 12h | WORD | reserve pool size |
| 14h | WORD | remaining space in reserve pool |
| 16h | WORD | required free memory |
| 18h | WORD | total cache sectors |
| 1Ah | WORD | dirty cache sectors |
| 1Ch | BYTE | trace buffer size |
| 1Dh | BYTE | reserved (padding) |

**17-11 Format of drive info [16-element array, one element]:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | DOS drive number |
| 01h | BYTE | BIOS drive number |
| 02h | BYTE | maximum sector number |
| 03h | BYTE | maximum head number |
| 04h | BYTE | read buffer size |
| 05h | BYTE | write buffer size |
| 06h | BYTE | last status |
| 07h | BYTE | flag: enabled |
| 08h | BYTE | flag: buffered write enabled |
| 09h | BYTE | flag: buffered read enabled |
| 0Ah | BYTE | flag: in use (drive info is valid) |
| 0Bh | BYTE | flag: cylinder flush |
| 0Ch | BYTE | reserved (padding) |
| 0Dh | BYTE | sectors per track |
| 0Eh | WORD | sector size |
| 10h | WORD | sectors assigned |
| 12h | WORD | dirty sectors |

| | | |
|---|---|---|
| 14h | WORD | reserved sectors |
| 16h | WORD | number of read errors |
| 18h | WORD | number of write errors |
| 1Ah | DWORD | "rio_count" |
| 1Eh | DWORD | number of cache misses |
| 22h | DWORD | "wio_count" |
| 26h | DWORD | "dio_count" |

---

### INT 13h Function 31h
### *RESERVE CONVENTIONAL MEMORY*

*QUICKCACHE II v4.20*
*(undocumented)*

**Registers at call:**
AH = 31h
BX = number of paragraphs of conventional memory to reserve for apps

**Return Registers:**
AX = status (0000h successful)

---

### INT 13h Function 32h
### *ENABLE CACHING FOR SPECIFIC DRIVE*

*QUICKCACHE II v4.20*
*(undocumented)*

**Registers at call:**
AH = 32h
AL = drive number (00h=A:)

**Return Registers:**
AX = status (0000h successful)

**See also:** Functions 22h and 33h

---

### INT 13h Function 33h
### *DISABLE CACHING FOR SPECIFIC DRIVE*

*QUICKCACHE II v4.20*
*(undocumented)*

**Registers at call:**
AH = 33h
AL = drive number (00h=A:)

**Return Registers:**
AX = status (0000h successful)

**See also:** Functions 22h and 32h

---

### INT 13h Function 34h
### *SECTOR LOCKING*

*QUICKCACHE II v4.20*
*(undocumented)*

**Purpose:** Allow specified disk sectors to be locked into the cache so that they are always present for quick access.

**Registers at call:**
AH = 34h
AL = function
    00h end sector locking/unlocking
    01h lock all accessed sectors into cache
    02h unlock all accessed sectors and discard from cache

**Return Registers:**
AX = status (0000h successful)

**See also:** Functions 20h and 35h

---

### INT 13h Function 35h
### *SET LOCK POOL SIZE*

*QUICKCACHE II v4.20*
*(undocumented)*

**Purpose:** Specify the number of cache sector buffers which may be dedicated to data locked into the cache.

**Registers at call:**
AH = 35h
BX = number of sectors in lock pool

**Return Registers:**
AX = status (0000h successful)

**See also:** Function 34h

---

### INT 13h Function 36h
### *SET TRACE BUFFER SIZE*

*QUICKCACHE II v4.20*
*(undocumented)*

Registers at call:
AH = 36h
AL = new size of trace buffer

**Return Registers:**
AX = status (0000h successful)

**Details:** This function is called with AL=05h during a Function 24h call.
**See also:** Function 24h

---

### INT 13h Function 37h

QUICKCACHE II v4.20 *(undocumented)*

#### *SET BUFFERED READS FOR SPECIFIC DRIVE*

**Registers at call:**
AH = 37h
AL = new state (00h disabled, else enabled)
DL = drive number (00h = A:)

**Return Registers:**
AX = status (0000h successful)

**See also:** Functions 2Dh and 38h

---

### INT 13h Function 38h

QUICKCACHE II v4.20 *(undocumented)*

#### *SET BUFFERED WRITES FOR SPECIFIC DRIVE*

**Registers at call:**
AH = 38h
AL = new state (00h disabled, else enabled)
DL = drive number (00h = A:)

**Return Registers:**
AX = status (0000h successful)

**See also:** Functions 2Ch and 37h

---

### INT 13h Function 39h

QUICKCACHE II v4.20 *(undocumented)*

#### *SET READ BUFFER SIZE FOR SPECIFIC DRIVE*

**Registers at call:**
AH = 39h
AL = new size of read buffer
DL = drive number (00h = A:)

**Return Registers:**
AX = status (0000h successful)

**See also:** Functions 2Ah and 3Ah

---

### INT 13h Function 3Ah

QUICKCACHE II v4.20 *(undocumented)*

#### *SET WRITE BUFFER SIZE FOR SPECIFIC DRIVE*

**Registers at call:**
AH = 3Ah
AL = new size of write buffer
DL = drive number (00h = A:)

**Return Registers:**
AX = status (0000h successful)

**See also:** Functions 2Ah and 39h

---

### INT 13h Function 3Bh

QUICKCACHE II v4.20 *(undocumented)*

#### *ENABLE/DISABLE UNKNOWN FUNCTIONALITY*

**Registers at call:**
AH = 3Bh
AL = new state of *unknown* (01h enabled, else disabled)

**Return Registers:**
AX = status (0000h successful)

**Details:** This function is affected by the flag reported at offset 05h of the system info returned by Function 30h, and sets the flag at offset 06h.
**See also:** Functions 30h and 3Ch

---

### INT 13h Function 3Ch

QUICKCACHE II v4.20 *(undocumented)*

#### *ENABLE/DISABLE UNKNOWN FUNCTIONALITY*

**Registers at call:**
AH = 3Ch
AL = new state of *unknown* (01h enabled, else disabled)

**Return Registers:**
AX = status (0000h successful)

**Details:** This function is affected by the flag reported at offset 05h of the system info returned by Function 30h, and sets the flag at offset 07h.
**See also:** Functions 30h and 3Bh

---

### INT 13h Function 3Dh
### *ENABLE/DISABLE CYLINDER FLUSH FOR DRIVE*

*QUICKCACHE II v4.20*
*(undocumented)*

**Registers at call:**
AH = 3Dh
AL = new state (01h enabled, else disabled)
DL = drive number (00h = A:)

**Return Registers:**
AX = status (0000h successful)

---

### INT 13h Function 3Eh
### *SET SINGLE-SECTOR BONUS*

*QUICKCACHE II v4.20*
*(undocumented)*

**Purpose:** Specify the bonus score to give to single-sector transfers in order to keep those sectors in the cache longer.

**Registers at call:**
AH = 3Eh
AL = new value for bonus

**Return Registers:**
AX = status (0000h successful)

**See also:** Function 3Fh

---

### INT 13h Function 3Fh
### *SET BONUS THRESHOLD*

*QUICKCACHE II v4.20*
*(undocumented)*

**Registers at call:**
AH = 3Fh
AL = new value for bonus threshold

**Return Registers:**
AX = status (0000h successful)

**See also:** Function 3Eh

---

### INT 13h Function 40h
### *SET "sticky_max"*

*QUICKCACHE II v4.20*
*(undocumented)*

**Registers at call:**
AH = 40h
AL = new value for "sticky_max"

**Return Registers:**
AX = status (0000h successful)

---

### INT 13h Function 41h
### *SAVE/RESTORE UNKNOWN VALUES*

*QUICKCACHE II v4.20*
*(undocumented)*

**Registers at call:**
AH = 41h
AL = direction
    01h save to file
    else restore from file
ES:DI -> 1024-byte buffer for *unknown*

**Return Registers:**
AX = status (0000h successful, 8000h failed)

## SmartDrive

SmartDrive is the disk cache bundled with MS-DOS 5+ and MS Windows.

---

### INT 21h Function 4402h
### *IOCTL - GET CACHE STATUS*

*SMARTDRV.SYS*

**Registers at call:**
AX = 4402h
BX = file handle for device "SMARTAAR"
CX = number of bytes to read (min 28h)
DS:DX -> buffer for status record (see Table 17-12)

**Return Registers:**
CF clear if successful
    AX = number of bytes actually read
CF set on error
    AX = error code (01h,05h,06h,0Dh)
    (see Table 24-2)

**See also:** Function 4403h, INT 2Fh Function 4A10h Subfunction 0000h

### 17-12 Format of SMARTDRV status record:

| *Offset* | *Size* | *Description* | |
|---|---|---|---|
| 00h | BYTE | write-through flag (always 01h) | |
| 01h | BYTE | writes should be buffered (always 00h) | |
| 02h | BYTE | cache enabled if 01h | |
| 03h | BYTE | driver type (01h extended memory, 02h expanded) | |
| 04h | WORD | clock ticks between cache flushes (currently unused) | |
| 06h | BYTE | cache contains locked tracks if nonzero | |
| 07h | BYTE | flush cache on INT 19h reboot if nonzero | |
| 08h | BYTE | cache full track writes if nonzero | |
| 09h | BYTE | double buffering (for VDS) state (00h off, 01h on, 02h dynamic) | |
| 0Ah | DWORD | original INT 13h vector | |
| 0Eh | BYTE | minor version number | |
| 0Fh | BYTE | major version number | |
| 10h | WORD | unused | |
| 12h | WORD | sectors read | } may be scaled rather than absolute counts |
| 14h | WORD | sectors already in cache | |
| 16h | WORD | sectors already in track buffer | |
| 18h | BYTE | cache hit rate in percent | |
| 19h | BYTE | track buffer hit rate in percent | |
| 1Ah | WORD | total tracks in cache | |
| 1Ch | WORD | number of tracks in use | |
| 1Eh | WORD | number of locked tracks | |
| 20h | WORD | number of dirty tracks | |
| 22h | WORD | current cache size in 16K pages | |
| 24h | WORD | original (maximum) cache size in 16K pages | |
| 26h | WORD | minimum cache size in 16K pages | |
| 28h | DWORD | pointer to byte flag to increment for locking cache contents | |

---

## INT 21h Function 4403h

*SMARTDRV.SYS*

### *CACHE CONTROL*

**Registers at call:**

AX = 4403h
BX = handle for device "SMARTAAR"
CX = number of bytes to write
DS:DX -> SMARTDRV control block (see Table 17-13)

**Return Registers:**

CF clear if successful
AX = number of bytes actually written (0000h if control block too small for given command)
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**See also:** Function 4402h, INT 2Fh Function 4A10h Subfunction 0000h

### 17-13 Format of SMARTDRV control block:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | function code<br>00h flush cache<br>01h flush and discard cache<br>02h disable caching (flushes and discards cache first)<br>03h enable caching<br>04h control write caching<br>05h set flushing tick count<br>06h lock cache contents<br>07h unlock cache contents<br>08h set flush-on-reboot flag<br>09h unused |

| | | |
|---|---|---|
| | | 0Ah control full-track caching<br>0Bh reduce cache size<br>0Ch increase cache size<br>0Dh set INT 13h chain address |
| **---function 04h---** | | |
| 01h | BYTE | write caching control action<br>00h turn off write-through<br>01h turn on write-through<br>02h turn off write buffering (also flushes cache)<br>03h turn on write buffering (also flushes cache) |
| **---function 05h---** | | |
| 01h | WORD | number of timer ticks between cache flushes |
| **---function 08h---** | | |
| 01h | BYTE | new flush-on-reboot flag (00h off, 01h on) |
| **---function 0Ah---** | | |
| 01h | BYTE | full-track writes (00h not cached, 01h cached) |
| **---functions 0Bh,0Ch---** | | |
| 01h | WORD | number of 16K pages by which to increase/reduce cache size |
| **---function 0Dh---** | | |
| 01h | DWORD | new address to which to chain on INT 13h; the previous address is not preserved. |

## INT 2Fh Function 4A10h, Subfunction 0000h

*SMARTDRV v4.00+*

### *INSTALLATION CHECK AND HIT RATIOS*

| Registers at call: | Return Registers: |
|---|---|
| AX = 4A10h | AX = BABEh if installed |
| BX = 0000h | DX:BX = cache hits |
| CX = EBABh (v4.1+; see Note) | DI:SI = cache misses |
| | CX = *unknown* |
| | BP = version in BCD (4.10 = 0410h) |

**Details:** Most of the SMARTDRV API, including this call, is supported by PC-Cache v8.0. If DBLSPACE.BIN is installed but SMARTDRV has not yet been installed, unless CX=EBABh on entry, DBLSPACE.BIN displays the error message "Cannot run SMARTDrive 4.0 with DoubleSpace" and aborts the caller with INT 21h Function 4C00h.

SMARTDRV v3.x had a completely different API using IOCTL calls; see above.

**See also:** Function 4A10h Subfunctions 0001h, 0004h, 0005h, 0007h, and 1234h; INT 21h Functions 4402h and 4403h; DoubleSpace Function 4A11h Subfunction 0000h (Chapter 16)

## INT 2Fh Function 4A10h, Subfunction 0001h

*SMARTDRV v4.00+*

### *FLUSH BUFFERS*

| Registers at call: | Return Registers: |
|---|---|
| AX = 4A10h | none |
| BX = 0001h | |

**Details:** This function is also supported by PC-Cache v8.0.

**See also:** Function 4A10h Subfunctions 0000h and 0002h

## INT 2Fh Function 4A10h, Subfunction 0002h

*SMARTDRV v4.00+*

### *RESET CACHE*

| Registers at call: | Return Registers: |
|---|---|
| AX = 4A10h | none |
| BX = 0002h | |

**Details:** This function is also supported by PC-Cache v8.0.

**See also:** Function 4A10h Subfunctions 0000h and 0001h

### INT 2Fh Function 4A10h, Subfunction 0003h

*SMARTDRV v4.00+*

### *STATUS*

**Registers at call:**

AX = 4A10h
BX = 0003h
BP = drive # (0=A, 1=B, etc.)
DL = subfunction
  00h only get information
  01h turn on read cache
  02h turn off read cache
  03h turn on write cache
  04h turn off write cache

**Return Registers:**

AX = BABEh if OK
DL = status
  bit 7 not cached
  bit 6 write-through
  bit 5 *unknown*
  bits 0-4 drive # (0=A, 1=B...)
DL = FFh if drive does not exist

**Details:** If the read cache is off, reads will not be cached, but writes will continue to be cached if the write-cache is enabled. This function is also supported by PC-Cache v8.0.

**See also:** Function 4A10h Subfunction 0000h

### INT 2Fh Function 4A10h, Subfunction 0004h

*SMARTDRV v4.00+*

### *GET CACHE SIZE*

**Registers at call:**

AX = 4A10h
BX = 0004h

**Return Registers:**

AX = *current size in elements*
BX = largest number of elements
CX = size of elements in bytes
DX = number of elements under Windows

**Details:** This function is also supported by PC-Cache v8.0.

**See also:** Function 4A10h Subfunctions 0000h, 0003h, and 0005h

### INT 2Fh Function 4A10h, Subfunction 0005h

*SMARTDRV v4.00+*

### *GET DOUBLE-BUFFER STATUS*

**Purpose:** Determine whether data transfers to the specified drive are double buffered to avoid address translation or DMA boundary problems.

**Registers at call:**

AX = 4A10h
BX = 0005h
BP = drive number (0=A, 1=B...)

**Return Registers:**

AX = BABEh if double-buffered
  ES:DI -> *unknown*

**See also:** Function 4A10h Subfunctions 0000h, 0003h, and 0006h

### INT 2Fh Function 4A10h, Subfunction 0006h

*SMARTDRV v4.00+*
*Callout (undocumented)*

### *CHECK IF DRIVE CACHEABLE*

**Registers at call:**

AX = 4A10h
BX = 0006h
CL = drive number (01h = A:)

**Return Registers:**

AX = 0006h if drive should not be cached by SMARTDRV

**Details:** SMARTDRV calls this function at startup to determine whether it should cache a particular drive.

**See also:** Function 4A10h Subfunction 0000h

### INT 2Fh Function 4A10h, Subfunction 0007h

*SMARTDRV v4.00+*

### *GET DEVICE DRIVER FOR DRIVE*

**Registers at call:**

AX = 4A10h
BX = 0007h
BP = drive number

**Return Registers:**

DL = *unknown*
ES:DI -> device driver header for drive

**Details:** This function is also supported by PC-Cache v8.0.
**See also:** Function 4A10h Subfunction 0000h, DoubleSpace Function 4A11h Subfunctions 0003h and 0004h (Chapter 16)

---

### INT 2Fh Function 4A10h, Subfunction 000Ah

*SMARTDRV v4.00+*

*GET TABLE POINTER*

**Registers at call:**
AX = 4A10h
BX = 000Ah

**Return Registers:**
ES:BX -> table of about 10 bytes or 5 words.
*Apparently words@S pointing to memory addresses containing info* (see Table 17-14)

**Details:** This function is also supported by PC-Cache v8.0.
**See also:** Function 4A10h Subfunction 0000h

**17-14 Format of data table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 8 BYTEs | *unknown* |
| 08h | WORD | offset of WORD containing number of elements in cache |

---

### INT 2Fh Function 4A10h, Subfunction 1234h

*SMARTDRV v4.00+*

***SIGNAL SERIOUS ERROR***

**Purpose:** Pops up a message box saying that a serious error occurred and to hit R to retry.

**Registers at call:**
AX = 4A10h
BX = 1234h

**Return Registers:**
none

**Details:** This function is also supported by PC-Cache v8.0.
**See also:** Function 4A10h Subfunction 0000h

## Super PC-Kwik

Super PC-Kwik is a commercial disk cache by PC-Kwik Corporation. A number of other manufacturer's caches are really OEM versions of Super PC-Kwik, among them Central Point's PC-Cache (versions 5.x only) and Qualitas' Qcache (version 4.00). PC-Cache versions 5.1 and 5.5 correspond to PC-Kwik versions 3.20 and 3.27, while Qcache corresponds to PC-Kwik version 4.00.

---

### INT 13h Functions 75h and 76h

*UNKNOWN*

***UNDETERMINED FUNCTIONS***

**Registers at call:**
AH = 75h and 76h
*others, if any, unknown*

**Return Registers:**
AH = *unknown*
*others, if any, unknown*

**Details:** These two functions are intercepted by PC-Cache (v5.1 only), and thus may also be intercepted by Super PC-Kwik version 3.20.

---

### INT 13h Function 81h

*Super PC-Kwik v3.20+*

***UNDETERMINED FUNCTION***

**Registers at call:**
AH = 81h
SI = 4358h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** PC Tools PC-Cache 5.x and Qualitas Qcache 4.00 are OEM versions of Super PC-Kwik, and thus support this call; however, this function returns immediately in PC-Cache v5.x.

---

### INT 13h Function 82h

*Super PC-Kwik v3.20+*

***UNDETERMINED FUNCTION***

**Registers at call:**
AH = 82h

SI = 4358h
*others, if any, unknown*

**Return Registers:**
AL = *unknown*

**Details:** PC Tools PC-Cache 5.x and Qualitas Qcache 4.00 are OEM versions of Super PC-Kwik, and thus support this call.
**See also:** Function 84h

---

### INT 13h Function 83h
*Super PC-Kwik v3.20+*

***UNDETERMINED FUNCTION***

**Registers at call:**
AH = 83h
SI = 4358h
AL = *unknown*
ES:BX -> *unknown*
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** PC Tools PC-Cache 5.x and Qualitas Qcache 4.00 are OEM versions of Super PC-Kwik, and thus support this call.
**See also:** Function 85h

---

### INT 13h Function 84h
*Super PC-Kwik v3.20+*

***UNDETERMINED FUNCTION***

**Registers at call:**
AH = 84h
SI = 4358h
AL = *unknown*
*others, if any, unknown*

**Return Registers:**
AL = *unknown*

**Details:** PC Tools PC-Cache 5.x and Qualitas Qcache 4.00 are OEM versions of Super PC-Kwik, and thus support this call.
**See also:** Function 82h

---

### INT 13h Function 85h
*Super PC-Kwik v3.20+*

***UNDETERMINED FUNCTION***

**Registers at call:**
AH = 85h
SI = 4358h
AL = *unknown*
DL = *unknown*
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** PC Tools PC-Cache 5.x and Qualitas Qcache 4.00 are OEM versions of Super PC-Kwik, and thus support this call.
**See also:** Function 83h

---

### INT 13h Function 86h
*Super PC-Kwik v4.00+*

***UNDETERMINED FUNCTION***

**Registers at call:**
AH = 86h
SI = 4358h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Qualitas Qcache v4.00 is an OEM version of Super PC-Kwik v4.00, and thus supports this call.

---

### INT 13h Function 87h
*Super PC-Kwik v4.00+*

***UNDETERMINED FUNCTION***

**Registers at call:**
AH = 87h

SI = 4358h
*others, if any, unknown*

**Return Registers:**
AH = *status* (00h)
CX = *unknown*
DX = *unknown* (0000h)

**Details:** Qualitas Qcache v4.00 is an OEM version of Super PC-Kwik v4.00, and thus supports this call.

---

### INT 13h Function 88h
### *UNDETERMINED FUNCTION*

*Super PC-Kwik v4.00+*

**Registers at call:**
AH = 88h
SI = 4358h
*others, if any, unknown*

**Return Registers:**
AH = *status* (00h)
CX = *unknown*
DX = *unknown* (0000h)

**Details:** Qualitas Qcache v4.00 is an OEM version of Super PC-Kwik v4.00, and thus supports this call.

---

### INT 13h Functions 89h and 8Ah
### *UNDETERMINED FUNCTIONS*

*Super PC-Kwik v5.10+*

**Registers at call:**
AH = 89h and 8Ah
SI = 4358h
*others, if any, unknown*

**Return Registers:**
*unknown*

---

### INT 13h Function 92h
### *UNDETERMINED FUNCTION*

*Super PC-Kwik v5.10+*

**Registers at call:**
AH = 92h
SI = 4358h
*others, if any, unknown*

**Return Registers:**
AH = *status* (00h)
DL = *unknown*

**See also:** Function 93h

---

### INT 13h Function 93h
### *UNDETERMINED FUNCTION*

*Super PC-Kwik v5.10+*

**Registers at call:**
AH = 93h
SI = 4358h
*others, if any, unknown*

**Return Registers:**
AH = *status* (00h)
AL = *unknown*

**See also:** Function 92h

---

### INT 13h Function 94h
### *UNDETERMINED FUNCTION*

*Super PC-Kwik v5.10+*

**Registers at call:**
AH = 94h
SI = 4358h
*others, if any, unknown*

**Return Registers:**
*unknown*

---

### INT 13h Function 95h
### *UNDETERMINED FUNCTION*

*Super PC-Kwik v5.10+*

**Registers at call:**
AH = 95h
SI = 4358h
*others, if any, unknown*

**Return Registers:**
AH = *status* (00h)
DX = *unknown*

### INT 13h Function 96h
### *UNDETERMINED FUNCTION*

*Super PC-Kwik v5.10+*

**Registers at call:**
AH = 96h
SI = 4358h
AL = *unknown* (01h)
BX = *unknown* (0790h)
DL = *unknown*

**Return Registers:**
AH = *status* (00h)
DX = *unknown*

### INT 13h Functions 97h to 9Bh
### *UNDETERMINED FUNCTIONS*

*Super PC-Kwik v5.10+*

**Registers at call:**
AH = 97h to 9Bh
SI = 4358h
*others, if any, unknown*

**Return Registers:**
*unknown*

### INT 13h Functions 9Ch and 9Dh
### *UNDETERMINED FUNCTIONS*

*Super PC-Kwik v5.10+*

**Registers at call:**
AH = 9Ch and 9Dh
SI = 4358h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Functions 9Ch and 9Dh are the only ones which are fully reentrant; all other PC-Kwik API calls (INT 13h/81h-B0h) return AX=0200h and CF clear if a previous call is still in progress.

### INT 13h Function A0h
### *GET RESIDENT CODE SEGMENT*

*Super PC-Kwik v3.20+*

**Registers at call:**
AH = A0h
SI = 4358h

**Return Registers:**
AX = segment of resident code

**Details:** PC Tools PC-Cache 5.x and Qualitas Qcache 4.00 are OEM versions of Super PC-Kwik, and thus support this call.

**See also:** PC-Cache INT 16h Function FFA5h Subfunction 1111h (above)

### INT 13h Function A1h
### *FLUSH CACHE*

*Super PC-Kwik v3.20+*

**Registers at call:**
AH = A1h
SI = 4358h

**Return Registers:**
CF clear
AH = 00h (v5.10)

**Details:** PC Tools PC-Cache 5.x and Qualitas Qcache 4.00 are OEM versions of Super PC-Kwik, and thus support this call.

**See also:** INT 16h Function FFA5h Subfunction FFFFh

### INT 13h Function A2h
### *UNDETERMINED FUNCTION*

*Super PC-Kwik v3.20+*

**Registers at call:**
AH = A2h
SI = 4358h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** PC Tools PC-Cache 5.x and Qualitas Qcache 4.00 are OEM versions of Super PC-Kwik, and thus support this call.

## INT 13h Function A3h

*Super PC-Kwik v5.10+ (undocumented)*

### *DISABLE CACHE*

**Registers at call:**
AH = A3h
SI = 4358h

**Return Registers:**
CF clear

**See also:** Function A4h

---

## INT 13h Function A4h

*Super PC-Kwik v5.10+ (undocumented)*

### *ENABLE CACHE*

**Registers at call:**
AH = A4h
SI = 4358h

**Return Registers:**
CF clear

**See also:** Function A3h

---

## INT 13h Function A5h

*Super PC-Kwik v5.10+ Callout (undocumented)*

### *PROGRAM TERMINATION NOTIFICATION*

**Registers at call:**
AH = A5h
SI = 4358h

**Return Registers:**
AX = *unknown*
SI = *unknown*

**Details:** This function is called and used internally by Super PC-Kwik when a program terminates via INT 21h Function 00h, INT 21h Function 31h, or INT 21h Function 4Ch.

This call is not supported by Qualitas Qcache 4.00.

**See also:** Functions A6h and A9h; MS-DOS INT 21h Functions 00h, 31h, and 4Ch (Chapter 24)

---

## INT 13h Function A6h

*Super PC-Kwik v5.10+ Callout (undocumented)*

### *PROGRAM LOAD NOTIFICATION*

**Registers at call:**
AH = A6h
SI = 4358h
DS:DX -> ASCIZ program name
ES:BX -> EXEC data block (see INT 21h Function 4Bh in Chapter 24)

**Return Registers:**
*unknown*

**Details:** This function is called and used internally by Super PC-Kwik when a program is loaded with INT 21h Function 4B00h.

**See also:** Functions A5h and A9h, MS-DOS INT 21h Function 4Bh (Chapter 24)

---

## INT 13h Function A7h

*Super PC-Kwik 5.1 Callout (undocumented)*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AH = A7h
SI = 4358h

**Return Registers:**
*unknown*

**Details:** This function is called and used internally by Super PC-Kwik on some INT 21h calls.

**See also:** Functions A5h, A6h, and A8h

---

## INT 13h Function A8h

*Super PC-Kwik 5.1 Callout (undocumented)*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AH = A8h
SI = 4358h

**Return Registers:**
*unknown*

**Details:** This function is called and used internally by Super PC-Kwik on some INT 21h calls.

**See also:** Functions A5h, A6h, and A7h

---

**INT 13h Function A9h** *Super PC-Kwik 5.1*
***EXITCODE RETRIEVAL NOTIFICATION*** *Callout (undocumented)*

**Registers at call:**
AH = A9h
SI = 4358h

**Return Registers:**
*unknown*

**Details:** This function is called and used internally by Super PC-Kwik when an application issues INT 21h Function 4Dh.

**See also:** Functions A5h and A6h, MS-DOS INT 21h Function 4Dh (Chapter 24)

---

**INT 13h Functions AAh to ADh** *Super PC-Kwik v4.0+*
***UNDETERMINED FUNCTIONS***

**Registers at call:**
AH = AAh to ADh
SI = 4358h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Qualitas Qcache is an OEM version of Super PC-Kwik, and thus supports these calls.

**Conflicts:** Priam Hard Disk Controller Function ADh (Chapter 10)

---

**INT 13h Function AEh** *Super PC-Kwik v5.10+*
***UNDETERMINED FUNCTION***

**Registers at call:**
AH = AEh
SI = 4358h
*others, if any, unknown*

**Return Registers:**
*unknown*

---

**INT 13h Function B0h** *Super PC-Kwik v3.20+*
***UNDETERMINED FUNCTION***

**Registers at call:**
AH = B0h
SI = 4358h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** PC Tools PC-Cache 5.x is an OEM version of Super PC-Kwik, and thus supports this call; Qualitas Qcache does not support it.

---

**INT 21h Function 2Bh, Subfunction 4358h** *Super PC-Kwik v3.20+*
***INSTALLATION CHECK***

**Registers at call:**
AH = 2Bh
CX = 4358h ('CX')

**Return Registers:**
AL = FFh if PC-Kwik or OEM version not installed
AL = 00h if installed
  CF clear
  CX = 6378h ('cx')
  BX = *unknown*
  DX = version (DH = major version, DL = binary minor version)

**Details:** PC Tools PC-Cache v5.x and Qualitas Qcache v4.00 are OEM versions of Super PC-Kwik, and thus support this call.

**See also:** INT 13h Functions A0h and B0h, PC-Cache INT 16h Function FFA5h Subfunction 1111h (above)

---

**INT 25h Function FFh** *PC-CACHE.SYS*
***INSTALLATION CHECK***

**Registers at call:**
AL = FFh
SI = 4358h

**Return Registers:**
SI = 6378h if installed

CX = segment of device driver PC-CACHE.SYS
DX = version (major in DH, minor in DL)

**Program:** PC-CACHE.SYS is a small device driver used by PC-Cache v5.x to obtain access to certain disk drivers for devices such as Bernoulli drives.

**See also:** INT 13h Function A0h, MS-DOS INT 25h (Chapter 24)

---

### INT 2Fh Function CCh
### *API*

*PC-Kwik Programs (undocumented)*

**Registers at call:**
AH = CCh
SI = 5453h or 7473h
AL = function number (at least
  00h,01h,04h,05h,07h,80h)
DL = program identifier
    00h all installed PowerPak programs
    01h SUPERPCK/SUPER/QCACHE v4.00+ (disk cache)
    02h PCKSPL (print spooler)
    04h PCKSCRN (screen accelerator)
    05h PCKKEY (keyboard enhancer)
    06h PCKRAMD (RAM disk)
    08h PCKWIK.SYS
    10h PCKCDROM (CD-ROM support for SUPERPCK)

**Return Registers:**
SI = 7473h
other registers vary by function

**Details:** Qualitas Qcache v4.00 is an OEM version of Super PC-Kwik v4.00, and thus supports this call.

**Conflicts:** none, but see Table 2-2

**See also:** INT 21h Function 2Bh Subfunction 4358h

## Miscellaneous

---

### INT 13h Function 1Dh
### *CACHE STATUS*

*IBMCACHE.SYS*

**Registers at call:**
AH = 1Dh
AL = subfunction
    01h get status record
        DL = *drive*
    02h set cache status
        ES:BX -> status record
        DL = *drive*

**Return Registers:**
(01h)
ES:BX -> status record
CF set on error
    AH = error code
(02h)
CF set on error

**17-15 Format of status record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | total number of read requests |
| 04h | DWORD | total number of hits |
| 08h | DWORD | number of physical disk reads |
| 0Ch | DWORD | total number of sectors requested by physical disk reads |
| 10h | 6 BYTEs | *unknown* |
| 16h | DWORD | pointer to start of error list (see Table 17-16) |
| 1Ah | DWORD | pointer to end of error list |
| 1Eh | WORD | *unknown* |

| | | |
|---|---|---|
| 20h | BYTE | using extended memory if nonzero |
| 21h | BYTE | *unknown* |
| 22h | 4 BYTEs | ASCII version number |
| 26h | WORD | cache size in KB |
| 28h | WORD | sectors per page |

**17-16 Format of error list:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | relative block address of bad page |
| 04h | BYTE | drive |
| 05h | BYTE | sector bit-map |
| 06h | WORD | next error |

### INT 16h Function 5758h
### *INSTALLATION CHECK*

*Netroom CACHECLK*
*(undocumented)*

**Registers at call:**
AX = 5758h
BX = 4858h ('HX')
DX = 4443h ('DC')
CX Ö 5758h

**Return Registers:**
BX = 6878h if installed
CX = 6463h if installed
AX = code segment of TSR
CX = *internal version* (v3.00 returns 0100h)

**Program:** CACHECLK is a "cloaked" disk cache included with Netroom.

**Details:** If CX=5758h on entry, CACHECLK returns with all registers unchanged. The cache statistics are located early in the segment pointed at by AX on return.

### INT 21h Function 4404h
### *GET DATA RECORD*

*COMBI-disk v1.13*

**Registers at call:**
AX = 4404h
BL = drive number (00h = default, 01h = A:, etc)
CX = (length of data packet)
DS:DX -> buffer for data packet (see Table 17-17)

**Return Registers:**
CF clear if successful
AX = number of bytes actually transferred
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Program:** COMBI-disk is a shareware combination cache and RAMdisk sharing a single pool of memory by Vadim V. Vlasov.

**Details:** The installation check consists of scanning the valid drives for one which returns a correctly-sized data packet with the proper signature in the first field.

**Conflicts:** xDISK (Chapter 10), Stacker (Chapter 16), DoubleDisk (Chapter 16), DoubleSpace (Chapter 16)

**See also:** Function 4405h

**17-17 Format of data packet:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 6 BYTEs | ASCIZ signature "COMBI" |
| 06h | WORD | version (high byte = major, low = decimal minor version) |
| 08h | BYTE | current options (see Table 17-19) |
| 09h | BYTE | sectors per allocation block |
| 0Ah | WORD | maximum buffer in KB |
| 0Ch | WORD | current buffer in KB (less than max if XMS memory being lent) |
| 0Eh | WORD | total number of allocation blocks |
| 10h | WORD | current number of allocation blocks |
| 12h | WORD | number of blocks being used by RAM disk |
| 14h | WORD | number of blocks being used by cache or unused |

| | | |
|---|---|---|
| 16h | WORD | number of dirty cache blocks |
| 18h | WORD | number of blocks which could not be written out due to errors |
| 1Ah | WORD | total number of read requests |
| 1Ch | WORD | total number of sectors read |
| 1Eh | WORD | number of BIOS read requests (cache misses) |
| 20h | WORD | number of sectors read via BIOS (cache misses) |
| 22h | WORD | total number of write requests |
| 24h | WORD | total number of sectors written |
| 26h | WORD | number of BIOS write requests |
| 28h | WORD | number of sectors written via BIOS |
| 2Ah | WORD | number of RAM disk read requests |
| 2Ch | WORD | number of sectors read from RAM disk |
| 2Eh | WORD | number of RAM disk write requests |
| 30h | WORD | number of sectors written to RAM disk |

## INT 21h Function 4405h

*COMBI-disk v1.13*

### *CONTROL COMBI-disk*

**Registers at call:**
AX = 4405h
BL = drive number (00h = default, 01h = A:, etc) for RAM disk
CX = number of bytes to write
DS:DX -> buffer containing command packet (see Table 17-18)

**Return Registers:**
CF clear if successful
  AX = number of bytes actually transferred
CF set on error
  AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Conflicts:** xDISK (Chapter 10), ST-01 SCSI.SYS (Chapter 11), DoubleSpace (Chapter 16)
**See also:** Function 4404h

**17-18 Format of command packet:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | version |
| 02h | BYTE | command code<br>80h flush cache<br>81h change options byte<br>82h shrink memory<br>83h expand memory<br>84h get dirty block information<br>85h reset write errors<br>86h reset counters |
| ---**command code 81h**--- | | |
| 03h | BYTE | new options byte (see Table 17-19) |
| ---**command code 82h**--- | | |
| 03h | WORD | number of KB to release |
| ---**command code 83h**--- | | |
| 03h | WORD | number of KB to expand |
| ---**command code 84h**--- | | |
| 03h | DWORD | -> buffer for block info (see Table 17-20) |
| ---**command code 85h**--- | | |
| 03h | DWORD | block ID |
| ---**command code 86h**--- | | |
| 03h | BYTE | which counters to reset<br>bit 0: hard disk read counts |

bit 1: hard disk write counts
bit 2: RAM disk read/write counts

**Note:** Multiple commands may be placed in a single packet by stringing together as many command/argument pairs as desired.

### 17-19 Bitfields for options byte:

| | |
|---|---|
| bit 0 | cache off |
| bit 1 | cache frozen |
| bit 2 | write caching enabled |
| bit 3 | delayed writing disabled |
| bit 5 | fix memory allocation (no XMS lending) |
| bit 6 | no 'sector not found' error |

### 17-20 Format of block info:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | block ID |
| 04h | BYTE | bitmask of valid sectors in block |
| 05h | BYTE | bitmask of dirty sectors in block |
| 06h | BYTE | last error returned by BIOS |
| 07h | BYTE | number of errors |

---

## INT 2Fh Function C64Ch
## *INSTALLATION CHECK*

*Smart Prompt*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = C64Ch | AX = 4CC6h if installed |
| BX = 5553h | BX = 5355h if installed |

**Program:** Smart Prompt (SMARTPMT) is a freeware TSR by Steve Gibson which forces an immediate flush of SmartDrive's (and compatible caches such as PC-Cache v8's) buffers on returning to the DOS prompt

**Conflicts:** none, but see Table 2-2

## CHAPTER 18

# Virtual DMA Specification

The Virtual DMA Specification was introduced with the Micro Channel models of IBM's PS/2 series to address problems inherent in protected-mode use of DMA and the presence of multiple bus masters independently performing DMA operatins. It contains, among others, functions to allocate buffers at fixed physical addresses and to lock address ranges to prevent multiple concurrent DMA operations on the same memory.

---

### INT 4Bh Function 8102h *VDS*

### *GET VERSION*

**Registers at call:**
AX = 8102h
DX = 0000h

**Return Registers:**
CF clear if successful
  AH = major version number
  AL = minor version number
  BX = product number (see Table 18-1)
  CX = product revision number
    always 0000h for QMAPS and HPMM.SYS
    always 0001h for Microsoft's EMM386.EXE v4.20-4.41
  SI:DI = maximum DMA buffer size
  DX = flags
    bit 0: PC/XT bus (DMA in first megabyte only)
    bit 1: physical buffer/remap region in first megabyte
    bit 2: automatic remap enabled
    bit 3: all memory is physically contiguous
    bits 4-15 reserved (zero)
CF set on error
  AL = error code (see Table 18-2)

**Details:** Bit 5 of 0040h:007Bh is supposed to be set if VDS is supported; this is apparently not always the case.

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Common Access Method (Chapter 11)

**See also:** DPMI INT 31h (Chapter 30), Cloaking INT 2Ch Function 002Bh (Chapter 32)

**18-1 VDS product numbers:**

| | |
|---|---|
| 0000h | Quadtel's QMAPS and Hewlett-Packard's HPMM.SYS |
| 0001h | Microsoft's EMM386.EXE |
| 0003h | Windows 3.x WIN386.EXE |
| 0EDCh | DR-DOS 6.0 EMM386.SYS |
| 4560h | ("E'") Qualitas' 386MAX |
| 4D43h | ("MC") V Communication's Memory Commander |

| | |
|---|---|
| 5145h | ("QE") Quarterdeck's QEMM-386 |
| 524Dh | ("RM") Helix's Netroom RM386 |

**18-2 VDS error codes:**

| | |
|---|---|
| 01h | region not in contiguous memory |
| 02h | region crossed a physical alignment boundary |
| 03h | unable to lock pages |
| 04h | no buffer available |
| 05h | region too large for buffer |
| 06h | buffer currently in use |
| 07h | invalid memory region |
| 08h | region was not locked |
| 09h | number of physical pages greater than table length |
| 0Ah | invalid buffer ID |
| 0Bh | copy out of buffer range |
| 0Ch | invalid DMA channel number |
| 0Dh | disable count overflow |
| 0Eh | disable count underflow |
| 0Fh | function not supported |
| 10h | reserved flag bits set in DX |

**18-3 Format of DMA descriptor structure (DDS):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | region size |
| 04h | DWORD | offset |
| 08h | WORD | segment/selector |
| 0Ah | WORD | buffer ID |
| 0Ch | DWORD | physical address |

**18-4 Format of Extended DMA descriptor structure (EDDS):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | region size |
| 04h | DWORD | offset |
| 08h | WORD | segment/selector |
| 0Ah | WORD | reserved |
| 0Ch | WORD | number available |
| 0Eh | WORD | number used |
| 10h | DWORD | region 0 physical address |
| 14h | DWORD | region 0 size in bytes |
| 18h | DWORD | region 1 physical address |
| 1Ch | DWORD | region 1 size in bytes |

**18-5 Format of Extended DMA descriptor structure (EDDS) with page table entries:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | region size |
| 04h | DWORD | offset |
| 08h | WORD | segment/selector |
| 0Ah | WORD | reserved |
| 0Ch | WORD | number available |
| 0Eh | WORD | number used |
| 10h | DWORD | page table entry 0 (same as 80386 page table entry) |
| 14h | DWORD | page table entry 1 |

**Note:** bits 1-11 of the page table entries should be zero; bit 0 set if page is present and locked

## INT 4Bh Function 8102h

### INT 4Bh Function 8103h

*VDS*

#### ***LOCK DMA REGION***

**Registers at call:**
AX = 8103h
DX = flags (see Table 18-6)
ES:DI -> DMA descriptor structure
(see Table 18-3)

**Return Registers:**
CF clear if successful
DDS physical address field filled in
DDS buffer ID field filled (0000h if no buffer allocated)
CF set on error
AL = error code (see Table 18-2)
DDS region size field filled wth maximum
contiguous length in bytes

**BUGS:** Windows 3.0 Enhanced mode does not correctly support automatic remapping or copying, nor does it return a correct code on error.

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Common Access Method (Chapter 11)

**See also:** Functions 8104h and 8105h

**18-6 Bitfields for flags:**

| | |
|---|---|
| bit 0 | reserved (zero) |
| bit 1 | data should be copied into buffer (ignored if bit 2 set) |
| bit 2 | buffer should not be allocated if region noncontiguous or crosses physical alignment boundary specified by bits 4-5 |
| bit 3 | don't attempt automatic remap |
| bit 4 | region must not cross 64K physical alignment boundary |
| bit 5 | region must not cross 128K physical alignment boundary |
| bits 6-15 | reserved (zero) |

### INT 4Bh Function 8104h

*VDS*

#### ***UNLOCK DMA REGION***

**Registers at call:**
AX = 8104h
DX = flags
bit 0: reserved (zero)
bit 1: data should be copied out of buffer
bits 2-15 reserved (zero)
ES:DI -> DMA descriptor structure (see Table 18-3) with region size, physical address, and buffer ID fields set

**Return Registers:**
CF clear if successful
DDS physical address field set
DDS buffer ID field set (0000h if no buffer allocated)
CF set on error
AL = error code (see Table 18-2)
DDS region size field filled wth maximum
contiguous length in bytes

**Details:** Windows 3.0 does not check whether the region extends beyond the end of a segment

**BUG:** Windows 3.0 in enhanced mode does not return a correct code on error.

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Common Access Method (Chapter 11)

**See also:** Functions 8103h and 8106h

### INT 4Bh Function 8105h

*VDS*

#### ***SCATTER/GATHER LOCK REGION***

**Registers at call:**
AX = 8105h
DX = flags
bits 0-5 reserved (zero)
bit 6: EDDS should be returned with page table entries
bit 7: only present pages should be locked (not-present pages receive entry of 0000h)

bits 8-15 reserved (zero)
ES:DI -> Extended DMA descriptor structure (see Tables 18-4 and 18-5)
region size, linear segment, linear offset, and number available fields set

**Return Registers:**
CF clear if successful
EDDS number used field set
if DX bit 6 set, lower 12 bits of BX = offset in first page
CF set on error
AL = error code (see Table 18-2)
EDDS region size field filled with max length in bytes that can be locked and described in the EDDS table

**BUG:** Windows 3.0 in enhanced mode may return zero instead of the physical page address for pages which were originally not present.
**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Common Access Method (Chapter 11)
**See also:** Functions 8103h and 8106h

---

### INT 4Bh Function 8106h

VDS

### *SCATTER/GATHER UNLOCK REGION*

**Registers at call:**
AX = 8106h
DX = flags
bits 0-5 reserved (zero)
bit 6: EDDS contains page table entries
bit 7: EDDS may contain not-present pages (entry = 0000h)
bits 8-15 reserved (zero)
ES:DI -> Extended DMA descriptor structure (see Tables 18-4 and 18-5) returned by Function 8105h

**Return Registers:**
CF clear if successful
CF set on error
AL = error code (see Table 18-2)

**Details:** According to the Microsoft version of the VDS specification, the actual scatter/gather list is ignored, while according to the IBM version of the specification, "the result of a LOCK operation" must be provided to this function.
**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Common Access Method (Chapter 11)
**See also:** Functions 8104h and 8105h

---

### INT 4Bh Function 8107h

VDS

### *REQUEST DMA BUFFER*

**Registers at call:**
AX = 8107h
DX = flags
bit 0: reserved (zero)
bit 1: data should be copied into buffer
bits 2-15 reserved (zero)
ES:DI -> DMA descriptor structure (see Table 18-3) with region size set (also region offset and region segment if DX bit 1 set)

**Return Registers:**
CF clear if successful
DDS physical address and buffer ID set
DDS region size filled with length of buffer
CF set on error
AL = error code (see Table 18-2)

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Common Access Method (Chapter 11)
**See also:** Function 8108h

---

### INT 4Bh Function 8108h

VDS

### *RELEASE DMA BUFFER*

**Registers at call:**
AX = 8108h
DX = flags

bit 0: reserved (zero)
bit 1: data should be copied out of buffer
bits 2-15 reserved (zero)
ES:DI -> DMA descriptor structure (see Table 18-3) with buffer ID set (also region size/region offset/segment if DX bit 1 set)

**Return Registers:**
CF clear if successful
CF set on error
AL = error code (see Table 18-2)

**BUG:** Under Windows 3.0 Enhanced mode, you must specify that data be copied for this function to work correctly.
**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Common Access Method (Chapter 11)
**See also:** Function 8107h

---

### INT 4Bh Function 8109h *VDS*
### *COPY INTO DMA BUFFER*

**Registers at call:**
AX = 8109h
DX = 0000h
ES:DI -> DMA descriptor structure (see Table 18-3) with buffer ID, region segment/offset, and region size fields set
BX:CX = starting offset into DMA buffer

**Return Registers:**
CF clear if successful
CF set on error
AL = error code (see Table 18-2)

**BUG:** Windows 3.0 Enhanced mode does not correctly interpret the copy count.
**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Common Access Method (Chapter 11)
**See also:** Function 810Ah

---

### INT 4Bh Function 810Ah *VDS*
### *COPY OUT OF DMA BUFFER*

**Registers at call:**
AX = 810Ah
DX = 0000h
ES:DI -> DMA descriptor structure (see Table 18-3) with buffer ID, region segment/offset, and region size fields set
BX:CX = starting offset into DMA buffer

**Return Registers:**
CF clear if successful
CF set on error
AL = error code (see Table 18-2)

**BUG:** Windows 3.0 Enhanced mode does not correctly interpret the copy count.
**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Common Access Method (Chapter 11)
**See also:** Function 8109h

---

### INT 4Bh Function 810Bh *VDS*
### *DISABLE DMA TRANSLATION*

**Registers at call:**
AX = 810Bh
BX = DMA channel number
DX = 0000h

**Return Registers:**
CF clear if successful
CF set on error
AL = error code (see Table 18-2)

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Common Access Method (Chapter 11)
**See also:** Function 810Ch

### INT 4Bh Function 810Ch
### *ENABLE DMA TRANSLATION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 810Ch | CF clear if successful |
| BX = DMA channel number | ZF set if disable count decremented to zero |
| DX = 0000h | CF set on error |
| | AL = error code (see Table 18-2) |

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Common Access Method (Chapter 11)
**See also:** Function 810Bh

## CHAPTER 19

# *Advanced Power Management Specification*

Power-managed personal computers, such as some battery-powered notebook computers, are designed to conserve power during periods of CPU inactivity or "idle time." Most such systems conform to the Advanced Power Management specification developed by Microsoft and Intel. This chapter describes the API for this specification.

---

### INT 15h Function 5300h

*APM*

***INSTALLATION CHECK***

**Registers at call:**
AX = 5300h
BX = 0000h (device ID of system BIOS)

**Return Registers:**
CF clear if successful
    AH = major version (BCD)
    AL = minor version (BCD)
    BX = 504Dh ("PM")
    CX = flags (see Table 19-2)
CF set on error
    AH = error code (86h) (see Table 19-1)

**19-1 APM error codes:**

| | |
|---|---|
| 01h | power management functionality disabled |
| 02h | interface connection already in effect |
| 03h | interface not connected |
| 04h | real-mode interface not connected |
| 05h | 16-bit protected-mode interface already connected |
| 06h | 16-bit protected-mode interface not supported |
| 07h | 32-bit protected-mode interface already connected |
| 08h | 32-bit protected-mode interface not supported |
| 09h | unrecognized device ID |
| 0Ah | invalid parameter value in CX |
| 0Bh-1Fh | reserved for other interface and general errors |
| 20h-3Fh | reserved for CPU errors |
| 40h-5Fh | reserved for device errors |
| 60h | can't enter requested state |
| 61h-7Fh | reserved for other system errors |
| 80h | no power management events pending |
| 81h-85h | reserved for other power management event errors |
| 86h | APM not present |
| 87h-9Fh | reserved for other power management event errors |

**19-2 APM flags:**

| | |
|---|---|
| bit 0 | 16-bit protected mode interface supported |
| bit 1 | 32-bit protected mode interface supported |
| bit 2 | CPU idle call reduces processor speed |
| bit 3 | BIOS power management disabled |
| bits 4-7 | reserved |

---

### INT 15h Function 5301h
### *CONNECT REAL-MODE INTERFACE*

*APM*

**Registers at call:**
AX = 5301h
BX = 0000h (device ID of system BIOS)

**Return Registers:**
CF clear if successful
CF set on error
AH = error code (02h,09h) (see Table 19-1)

**See also:** Functions 5302h, 5303h, and 5304h

---

### INT 15h Function 5302h
### *CONNECT 16-BIT PROTECTED-MODE INTERFACE*

*APM*
*Real Mode*

**Registers at call:**
AX = 5302h
BX = 0000h (device ID of system BIOS)

**Return Registers:**
CF clear if successful
AX = real-mode segment base address of protected-mode 16-bit code segment
BX = offset of entry point
CX = real-mode segment base address of protected-mode 16-bit data segment
CF set on error
AH = error code (05h,06h,09h) (see Table 19-1)

**Details:** The caller must initialize two consecutive descriptors with the returned segment base addresses; these descriptors must be valid whenever the protected-mode interface is called, and will have their limits arbitrarily set to 64K.

The protected mode interface is invoked by making a far call with the same register values as for INT 15h; it must be invoked while CPL=0, the code segment descriptor must have a DPL of 0, the stack must be in a 16-bit segment and have enough room for BIOS use and possible interrupts, and the current I/O permission bit map must allow access to the I/O ports used for power management.

Functions 00h-03h are not available from protected mode.

**See also:** Functions 5301h, 5303h, and 5304h

---

### INT 15h Function 5303h
### *CONNECT 32-BIT PROTECTED-MODE INTERFACE*

*APM*
*Real Mode*

**Registers at call:**
AX = 5303h
BX = 0000h (device ID of system BIOS)

**Return Registers:**
CF clear if successful
AX = real-mode segment base address of protected-mode 32-bit code segment
EBX = offset of entry point
CX = real-mode segment base address of protected-mode 16-bit code segment
DX = real-mode segment base address of protected-mode 16-bit data segment
CF set on error
AH = error code (07h,08h,09h) (see Table 19-1)

**Details:** The caller must initialize three consecutive descriptors with the returned segment base addresses for 32-bit code, 16-bit code, and 16-bit data, respectively; these descriptors must be valid whenever the protected-mode interface is called, and will have their limits arbitrarily set to 64K.

The protected mode interface is invoked by making a far call to the 32-bit code segment with the same register values as for INT 15h; it must be invoked while CPL=0, the code segment descriptor must have a DPL of 0, the stack must be in a 32-bit segment and have enough room for BIOS use and possible interrupts, and the current I/O permission bit map must allow access to the I/O ports used for power management.

Functions 00h-03h are not available from protected mode.

**See also:** Functions 5301h, 5302h, and 5304h

---

**INT 15h Function 5304h** *APM*

***DISCONNECT INTERFACE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5304h | CF clear if successful |
| BX = 0000h (device ID of system BIOS) | CF set on error<br>AH = error code (03h,09h) (see Table 19-1) |

**See also:** Functions 5301h, 5302h, and 5303h

---

**INT 15h Function 5305h** *APM*

***CPU IDLE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5305h | after system leaves idle state<br>CF clear |

**Details:** Call this function when the system is idle and should be suspended until the next system event or interrupt.

This function should not be called from within a hardware interrupt handler to avoid reentrance problems.

If an interrupt causes the system to resume normal processing, the interrupt may or may not have been handled when the BIOS returns from this call; thus, the caller should allow interrupts on return.

Interrupt handlers may not retain control if the BIOS allows interrupts while in idle mode even if they are able to determine that they were called from idle mode.

The caller should issue this call continuously in a loop until it needs to perform some processing of its own.

**See also:** Function 5306h, INT 2Fh Function 1680h (Chapter 43), DESQview Function 1000h (Chapter 44)

---

**INT 15h Function 5306h** *APM*

***CPU BUSY***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5306h | CF clear |

**Details:** This function may be called to ensure that the system runs at full speed even on systems where the BIOS is unable to recognize increased activity (especially if interrupts are hooked by other programs and not chained to the BIOS).

This call may be made even when the system is already running at full speed, but it will create unnecessary overhead.

This function should not be called from within a hardware interrupt handler to avoid reentrance problems.

**See also:** Function 5305h

---

**INT 15h Function 5307h** *APM*

***SET POWER STATE***

**Registers at call:**

AX = 5307h

BX = device ID (see table 19-3)

CX = system state ID
    0000h ready (not supported for device ID 0001h)
    0001h stand-by
    0002h suspend
    0003h off (not supported for device ID 0001h)
    0004h-FFFFh reserved

**Return Registers:**
CF clear if successful
CF set on error
    AH = error code (01h,09h,0Ah,60h) (see Table 19-1)

**Details:** This function should not be called from within a hardware interrupt handler to avoid reentrance problems.

**19-3 APM device IDs:**

| | |
|---|---|
| 0000h | system BIOS |
| 0001h | all devices for which the system BIOS manages power |
| 01xxh | display (01FFh for all attached display devices) |
| 02xxh | secondary storage (02FFh for all attached secondary storage devices) |
| 03xxh | parallel ports (03FFh for all attached parallel ports) |
| 04xxh | serial ports (04FFh for all attached serial ports) |
| 0500h-FFFFh | reserved |

---

### INT 15h Function 5307h, Subfunction 0001h

APM

### *SYSTEM STAND-BY*

**Registers at call:**
AX = 5307h
BX = 0001h
CX = 0001h

**Return Registers:**
CF clear

**Details:** This function puts the entire system into stand-by mode; it is normally called in response to a System Stand-by Request notification after any necessary processing, but may also be invoked at the caller's discretion.

This function should not be called from within a hardware interrupt handler to avoid reentrance problems.

The stand-by state is typically exited on an interrupt.

**See also:** Function 5307h Subfunction 0002h, Function 530Bh, Compaq Function 4280h (Chapter 7)

---

### INT 15h Function 5307h, Subfunction 0002h

APM

### *SUSPEND SYSTEM*

**Registers at call:**
AX = 5307h
BX = 0001h
CX = 0002h

**Return Registers:**
after system is resumed
CF clear

**Details:** This function puts the entire system into a low-power suspended state; it is normally called in response to a Suspend System Request notification after any necessary processing, but may also be invoked at the caller's discretion.

This function should not be called from within a hardware interrupt handler to avoid reentrance problems.

The caller may need to update its date and time values because the system could have been suspended for a long period of time.

**See also:** Function 5307h Subfunction 0001h, Function 530Bh

---

### INT 15h Function 5308h

APM

### *ENABLE/DISABLE POWER MANAGEMENT*

**Registers at call:**
AX = 5308h

BX = FFFFh
CX = new state
    0000h disabled
    0001h enabled

**Return Registers:**
CF clear if successful
CF set on error
    AH = error code (01h,09h,0Ah) (see Table 19-1)

**Details:** When power management is disabled, the system BIOS will not automatically power down devices, enter stand-by or suspended mode, or perform any power-saving actions in response to Function 5305h calls.

This function should not be called from within a hardware interrupt handler to avoid reentrance problems.

**See also:** Function 5309h

---

### INT 15h Function 5309h
### *RESTORE POWER-ON DEFAULTS*

*APM*

**Registers at call:**
AX = 5309h
BX = FFFFh

**Return Registers:**
CF clear if successful
CF set on error
    AH = error code (09h) (see Table 19-1)

**Details:** This function should not be called from within a hardware interrupt handler to avoid reentrance problems.

**See also:** Function 5308h

---

### INT 15h Function 530Ah
### *GET POWER STATUS*

*APM*

**Registers at call:**
AX = 530Ah
BX = 0001h

**Return Registers:**
CF clear if successful
    BH = AC line status
        00h off-line
        01h on-line
        FFh unknown
        other reserved
    BL = battery status
        00h high
        01h low
        02h critical
        03h charging
        FFh unknown
        other reserved
    CL = remaining battery life
        00h-64h (0-100) percentage of full charge
        FFh unknown
CF set on error
    AH = error code (09h) (see Table 19-1)

**Details:** This function should not be called from within a hardware interrupt handler to avoid reentrance problems.

---

### INT 15h Function 530Bh
### *GET POWER MANAGEMENT EVENT*

*APM*

**Registers at call:**
AX = 530Bh

**Return Registers:**
CF clear if successful
    BX = event code (see Table 19-4)
CF set on error
    AH = error code (03h,80h) (see Table 19-1)

**Details:** Although power management events are often asynchronous, notification will not be made until polled via this call to permit software to only receive event notification when it is prepared to process power management events; since these events are not very time-critical, it should be sufficient to poll once or twice per second.

The critical resume notification is made after the system resumes from an emergency suspension; normally, the system BIOS only notifies its partner that it wishes to suspend and relies on the partner to actually request the suspension, but no notification is made on an emergency suspension.

This function should not be called from within a hardware interrupt handler to avoid reentrance problems.

**See also:** Function 5307h Subfunctions 0000h to 0002h

**19-4 APM event codes:**

| | |
|---|---|
| 0001h | system stand-by request |
| 0002h | system suspend request |
| 0003h | normal resume system notification |
| 0004h | critical resume system notification |
| 0005h | battery low notification |

### INT 2Fh Function 4Ch
### *API*

*Advanced Power Management*
*(undocumented)*

**Registers at call:**
AH = 4Ch
AL = function
  00h version check
  01h suspend system requested
  FFh suspend/resume battery notification
*others, if any, unknown*

**Return Registers:**
*unknown*

### INT 2Fh Function 53h
### *APM EVENT BROADCASTING*

*POWER.EXE*
*(undocumented)*

**Registers at call:**
AH = 53h
AL = *event*
  05h CPU idle
  0Bh PM event broadcast API

**Return Registers:**
*unknown*

**Details:** This function is called by the MS Windows 3.1 POWER.DRV and hooked by the MS Mouse driver v8.20+ and PC-Cache v8.0.

**See also:** INT 15h Function 530Bh, Functions 54xxh, Mouse INT 33h Function 002Fh (Chapter 21)

### INT 2Fh Function 5400h
### *INSTALLATION CHECK*

*POWER.EXE*
*(undocumented)*

**Registers at call:**
AX = 5400h

**Return Registers:**
AX = POWER.EXE version (AH = major, AL = minor) if installed
BX = 504Dh ("PM")
CF clear

**Details:** This function is called by the MS Windows 3.1 POWER.DRV.

**Conflicts:** TeSseRact interface (Chapter 51)

**See also:** Function 53h; Functions 5401h, 5402h, 5481h, and 5482h

### INT 2Fh Function 5401h
### *GET/SET POWER STATUS*

*POWER.EXE*
*(undocumented)*

**Registers at call:**
AX = 5401h
BX = function
  0000h get status

**Return Registers:**
BL = power management status
  bit 0: CPU idle
  bit 1: *unknown*
*others, if any, unknown*

**Return Registers:**
AX = function status
  0000h success
  else error code

**Details:** This function is called by the MS Windows 3.1 POWER.DRV.

**Conflicts:** TeSseRact interface (Chapter 51)

**See also:** Function 53h; Functions 5400h, 5402h, and 5403h

---

### INT 2Fh Function 5402h
### *GET/SET IDLE DETECTION STRATEGY*

*POWER.EXE*
*(undocumented)*

**Registers at call:**
AX = 5402h
BH = subfunction
  00h get
  other set
    BL = detection strategy (00h-0Fh or FFh)

**Return Registers:**
BX = current/new detection strategy

**Conflicts:** TeSseRact interface (Chapter 51)

**See also:** Function 53h; Functions 5400h, 5401h, 5481h, and 5482h

---

### INT 2Fh Function 5403h
### *GET/SET ADVANCED POWER MANAGEMENT SETTING*

*POWER.EXE*
*(undocumented)*

**Registers at call:**
AX = 5403h
BX = new power management setting or 0000h to get current setting

**Return Registers:**
AX = status
  0000h successful
    BX = power management setting (see Table 19-5)
  other error code

**Conflicts:** TeSseRact interface (Chapter 51)

**See also:** Functions 5401h and 5480h

**19-5 Values for power management setting:**

| | |
|---|---|
| 0001h-0005h | "min" |
| 0006h | "reg" |
| 0007h-0008h | "max" |

---

### INT 2Fh Function 5480h
### *GET/SET UNKNOWN VALUE*

*POWER.EXE*
*(undocumented)*

**Registers at call:**
AX = 5480h
BX = direction
  0000h get
  other set
CX = size of buffer (at least 0010h)
DS:SI -> buffer

**Return Registers:**
AX = status
  0000h successful
  other error code

**Conflicts:** TeSseRact interface (Chapter 51)

**See also:** Functions 5400h, 5481h, and 548Fh

## INT 2Fh Function 5481h
### *GET STATISTICS*

*POWER.EXE*
*(undocumented)*

**Registers at call:**
AX = 5481h
BX = *unknown* (0000h or 0001h)
CX = length of buffer
DS:SI -> buffer for statistics (see Table 19-6)

**Return Registers:**
AX = status
0000h successful
else error code

**Conflicts:** TeSseRact interface (Chapter 51)
**See also:** Function 53h; Functions 5400h, 5480h, 5402h, and 5482h

**19-6 Format of statistics:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | total *unknown* |
| 04h | DWORD | *unknown* during which CPU was idle (divide by previous to get idle rate) |

## INT 2Fh Function 5482h
### *GET/SET APM POLLING FREQUENCY*

*POWER.EXE*
*(undocumented)*

**Registers at call:**
AX = 5482h
BX = new polling frequency or 0000h to get current frequency

**Return Registers:**
AX = 0000h (successful)
BX = current frequency if BX=0000h on entry

**Conflicts:** TeSseRact interface (Chapter 51)
**See also:** Function 53h; Functions 5400h, 5401h, 5480h, 5481h, and 548Fh

## INT 2Fh Function 548Fh
### *GET/SET* UNKNOWN VALUE

*POWER.EXE*
*(undocumented)*

**Registers at call:**
AX = 548Fh
BX = *unknown* or 0000h to get current *unknown*

**Return Registers:**
AX = 0000h (successful)
BX = current *unknown*`
CX = *unknown*

**Conflicts:** TeSseRact interface (Chapter 51)
**See also:** Functions 5400h, 5480h, and 5482h

## CHAPTER 20

# *Hewlett-Packard HP 95LX System Services*

---

**INT 5Fh Function 00h** *HP 95LX GRAPHICS PRIMITIVES*

***SET VIDEO MODE***

**Registers at call:**
AH = 00h
AL = video mode
  07h text, system manager compliant 20h 240x128 mono graphics, system manager compliant
  87h text, not system manager compliant
  A0h 240x128 mono graphics, not system manager compliant

**Return Registers:**
none

**Details:** The defaults after setting the mode to graphics are: logical origin (0,0), full-screen clip region, pen location (0,0), pen color 1, pixel replacement FORCE, line type and fill mask all bits set.

**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)

**See also:** HP 95LX INT 0Fh and INT 15h Function 4DD4h (Chapter 7), Standard Video INT 10h Function 00h (Chapter 8)

---

**INT 5Fh Function 01h** *HP 95LX GRAPHICS PRIMITIVES*

***SET FILL MASK***

**Registers at call:**
AH = 01h
ES:DI -> 8-byte fill mask

**Return Registers:**
none

**Details:** The fill mask represents an 8x8 pixel box and is repeated as necessary when drawing filled rectangles; it is always aligned with the byte boundaries of video memory, regardless of the actual boundaries of the rectangle.

**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)

**See also:** Function 02h

---

**INT 5Fh Function 02h** *HP 95LX GRAPHICS PRIMITIVES*

***GET CURRENT GRAPHICS INFORMATION***

**Registers at call:**
AH = 02h
ES:DI -> graphics info record
  (seeTable 20-1)

**Return Registers:**
DX:AX -> filled graphics info record (for return to
  high-level langs)

**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)

**See also:** Functions 03h and 04h

**20-1 Format of graphics info record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | current video mode |

| | | |
|---|---|---|
| 01h | BYTE | default video mode |
| 02h | WORD | display width in pixels |
| 04h | WORD | display height in pixels |
| 06h | WORD | current pen column |
| 08h | WORD | current pen row |
| 0Ah | WORD | current line type |
| 0Ch | WORD | current replacement rule |
| 0Eh | WORD | current pen color |
| 10h | WORD | current leftmost column of clip region |
| 12h | WORD | current rightmost column of clip region |
| 14h | WORD | current topmost row of clip region |
| 16h | WORD | current bottommost row of clip region |
| 18h | WORD | current column of logical origin |
| 1Ah | WORD | current row of logical origin |
| 1Ch | 8 BYTEs | current fill mask |

---

## INT 5Fh Function 03h

*HP 95LX GRAPHICS PRIMITIVES*

### *SET LOGICAL ORIGIN*

**Registers at call:**
AH = 03h
CX = column
DX = row

**Return Registers:**
none

**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)
**See also:** Function 04h

---

## INT 5Fh Function 04h

*HP 95LX GRAPHICS PRIMITIVES*

### *SET CLIP REGION*

**Registers at call:**
AH = 04h
CX = left-most column
DX = top-most row
SI = right-most column
DI = bottom-most row

**Return Registers:**
none

**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)
**See also:** Function 03h

---

## INT 5Fh Function 05h

*HP 95LX GRAPHICS PRIMITIVES*

### *DRAW RECTANGLE*

**Registers at call:**
AH = 05h
AL = fill type
    00h outline, using current line type and color
    01h solid, using current color
    02h pattern, using current fill mask and color
DX,CX = row,column of other corner of rectangle

**Return Registers:**
none

**Details:** The rectangle is drawn starting at the current pen position.
**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)
**See also:** Functions 01h, 06h, and 07h

---

**INT 5Fh Function 06h** *HP 95LX GRAPHICS PRIMITIVES*

***DRAW LINE***

**Registers at call:**
AH = 06h
DX,CX = row, column of end point

**Return Registers:**
none

**Details:** The line is drawn starting at the current pen position.

**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)

**See also:** Functions 05h and 07h

---

**INT 5Fh Function 07h** *HP 95LX GRAPHICS PRIMITIVES*

***PLOT POINT***

**Registers at call:**
AH = 07h
DX,CX = row,column of point

**Return Registers:**
none

**Details:** This function also sets the pen position to the specified point.

**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)

**See also:** Functions 06h, 08h, and 0Ch

---

**INT 5Fh Function 08h** *HP 95LX GRAPHICS PRIMITIVES*

***MOVE PEN***

**Registers at call:**
AH = 08h
DX,CX = row,column of new pen position

**Return Registers:**
none

**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)

**See also:** Functions 07h and 09h

---

**INT 5Fh Function 09h** *HP 95LX GRAPHICS PRIMITIVES*

***SET PEN COLOR***

**Registers at call:**
AH = 09h
AL = new color (00h = white, 01h = black)

**Return Registers:**
none

**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)

**See also:** Functions 08h, 0Ah, and 0Bh

---

**INT 5Fh Function 0Ah** *HP 95LX GRAPHICS PRIMITIVES*

***SET REPLACEMENT RULE***

**Registers at call:**
AH = 0Ah
AL = new replacement rule
  00h force
  01h AND
  02h OR
  03h XOR

**Return Registers:**
none

**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)

**See also:** Functions 01h, 09h, and 0Bh

---

**INT 5Fh Function 0Bh** *HP 95LX GRAPHICS PRIMITIVES*

***SET LINE TYPE***

**Registers at call:**
AH = 0Bh
CX = new line type

**Return Registers:**
none

**Details:** The line type specifies 16 bits that are repeated over and over while drawing the pixels of a line.

**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)

**See also:** Functions 09h and 0Ah

---

**INT 5Fh Function 0Ch** *HP 95LX GRAPHICS PRIMITIVES*

***GET PIXEL***

**Registers at call:**
AH = 0Ch
DX,CX = row,column of pixel to read

**Return Registers:**
AX = pixel color

**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)
**See also:** Function 07h

---

**INT 5Fh Function 0Dh** *HP 95LX GRAPHICS PRIMITIVES*

***GET IMAGE***

**Registers at call:**
AH = 0Dh
DX,CX = row,column of first corner
BP,SI = row,column of second corner ES:DI -> image buffer (see Table 20-2)

**Return Registers:**
none

**Details:** The specified corners are included in the saved image.
**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)
**See also:** Function 0Eh

**20-2 Format of image buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of planes (always 01h on HP 95LX) |
| 02h | WORD | number of bits/pixel (always 01h on HP 95LX) |
| 04h | WORD | image width in pixels |
| 06h | WORD | image height in pixels |
| 08h | N BYTEs | image data<br>requires (WIDTH+7)/8 * HEIGHT bytes |

---

**INT 5Fh Function 0Eh** *HP 95LX GRAPHICS PRIMITIVES*

***PUT IMAGE***

**Registers at call:**
AH = 0Eh
AL = replacement rule
  bit 2: invert image before applying rule
  bits 1-0: replacement rule
    00 force
    01 AND
    10 OR
    11 XOR
DX,CX = row,column of top left corner
ES:DI -> image buffer (see Table 20-2)

**Return Registers:**
none

**Details:** If the specified image does not fit completely on the screen, this call does nothing.
**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)
**See also:** Function 0Dh

---

**INT 5Fh Function 0Fh** *HP 95LX GRAPHICS PRIMITIVES*

***WRITE TEXT***

**Registers at call:**
AH = 0Fh
AL = rotate flag (if nonzero, rotate 90 degrees counter-clockwise)
DX,CX = row,column of first character's top left corner
ES:DI -> ASCIZ text

**Return Registers:**
none

**Conflicts:** DESQview relocated IRQ15 (Chapter 44), DoubleDOS relocated IRQ7 (Chapter 49)
**See also:** INT 60h Functions 0300h, 0F08h, and 1506h

---

### INT 60h Function 0100h
### *WAIT FOR EVENT*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0100h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to event record (see Table 20-4) |

**Return Registers:**
event record filled
STACK unchanged

**Details:** This call will timeout after about 500ms.
**Conflicts:** see Table 6-3
**See also:** INT 15h Function 4DD4h, Function 0101h, HP 95LX INT 61h, HP 95LX INT 62h

#### 20-3 System Manager event types:

| | |
|---|---|
| 00h | no events |
| 01h | keystroke available |
| 02h | Ctrl-Break |
| 03h | reactivation (always follows deactivation event) |
| 04h | about to deactivate (sleep)<br>next get-event call will not return until reactivated |
| 05h | forced application termination |
| 06h | 1-2-3 bridge service request (only given to 1-2-3) |
| 07h | request to grow |
| 08h | request to shrink |
| 09h | application's alarm expired |
| 0Ah | daily chance to set an alarm |
| 0Bh | system date or time has been changed |

#### 20-4 Format of event record:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | event type (see Table 20-3) |
| 02h | WORD | ASCII code page 850 translation of keystroke<br>or grow/shrink amount in paragraphs or 0000h if error<br>or alarm expiration data |
| 04h | BYTE | scan code from BIOS |
| 05h | BYTE | shift key states at time keystroke is retrieved |
| 06h | WORD | LICS translation of keystroke |
| 08h | BYTE | function key number (1-2-3 only) |
| 09h | DWORD | pointer to 1-2-3 bridge record (see Table 20-7)<br>or pointer to time change structure (see Table 20-5) |

**Note:** If the System Manager is awaiting the conclusion of a bridge service or grow/shrink call and the event type field is set to FFFFh on entry, the System Manager will resume.

#### 20-5 Format of time change structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | old year |
| 02h | BYTE | old month |
| 03h | BYTE | old date |
| 04h | BYTE | old day |
| 05h | BYTE | old hour |

| | | |
|---|---|---|
| 06h | BYTE | old minute |
| 07h | BYTE | old second |
| 08h | BYTE | old hundredth of a second |
| 09h | 9 BYTEs | new time in same format as old time |

---

### INT 60h Function 0101h
### *CHECK FOR EVENT*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0101h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to event record (see Table 20-4)

**Return Registers:**
event record filled
STACK unchanged

**Details:** This call returns immediately if no event is available.

**Conflicts:** see Table 6-3

**See also:** Function 0100h

---

### INT 60h Function 0102h
### *"SH_STATUS"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0102h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0104h
### *LOTUS 1-2-3 BRIDGE SERVICES*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0104h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to bridge record (see Table 20-7)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**20-6 Bridge Services function codes:**

| | |
|---|---|
| 00h | test |
| 01h | get range |
| 02h | "GETRANGE_ADDR" |
| 03h | "SETRANGE_ADDR" |
| 04h | "GETRANGE_DATA" |
| 05h | "SETRANGE_DATA" |
| 06h | recalculate |
| 07h | get cursor |
| 08h | set cursor |
| 09h | redisplay |
| 0Ah | cell type |
| 0Bh | "CALCTYPE" |

**20-7 Format of bridge record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | function code (see Table 20-6) |
| 02h | WORD | return code from 1-2-3 |
| 04h | 16 BYTEs | ASCII range name |
| 14h | WORD | start column of range |
| 16h | WORD | start row of range |
| 18h | WORD | end column of range |
| 1Ah | WORD | end row of range |
| 1Ch | WORD | order in which data is placed in buffer |
| 1Eh | WORD | buffer size |
| 20h | WORD | offset within bridge record's segment of buffer for cell data |

---

### INT 60h Function 0105h
### *FLUSH KEYBOARD BUFFER*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0105h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0106h
### *YIELD CPU*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0106h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**See also:** TopView INT 15h Function 1000h (Chapter 44), MS Windows INT 2Fh Function 1680h (Chapter 43)

---

### INT 60h Function 0107h Function 0107h
### *"NO_FINI" - REFUSE TERMINATION REQUEST*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0107h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**See also:** INT 61h and INT 62h below, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

### INT 60h Function 0200h
### *SETUP MENU*

*HP 95LX System Manager*
*(partially doc.)*

**Purpose:** Prepare a menu for display.

**Registers at call:**
DI = 0200h
STACK: 2 WORDs unused dummies (for calls from high level

languages)
DWORD pointer to menu data (see Table 20-8)
DWORD pointer to *unknown*
WORD *number of items on menu*
WORD *unknown*
DWORD pointer to *unknown*
WORD *unknown*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0201h, 0203h, and 0205h; HP 95LX INT 15h Function 4DD4h (Chapter 7)

**20-8 Format of menu data:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 80 BYTEs | first line of menu text |
| 50h | 80 BYTEs | second line of menu text |
| A0h | 80 BYTEs | third line of menu text |
| F0h | WORD | number of keywords |
| F2h | WORD | index of currently highlighted keyword or FFFFh |
| F4h | WORD | single prompt on top line if nonzero |
| F6h | 20 BYTEs | which line each of 20 keywords is located on |
| 10Ah | 20 BYTEs | offset of each of 20 keywords within its line |
| 11Eh | 20 BYTEs | length of each of 20 keywords |
| 132h | 20 BYTEs | first letter of each of 20 keywords |
| 146h | 20 WORDs | offsets of long prompts for each of 20 keywords |

### INT 60h Function 0201h
### *DISPLAY OR REDISPLAY MENU*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0201h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to menu data (see Table 20-8)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0200h, 0202h, and 0206h

### INT 60h Function 0202h
### *"MENU_ON" - ENABLE PROCESSING OF MENU*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0202h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to menu data (see Table 20-8)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0200h, 0201h, and 0203h

### INT 60h Function 0203h
### *REMOVE MENU*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0203h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to menu data (see Table 20-8) |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0201h, 0202h, 0204h, and 0208h

### INT 60h Function 0204h
### *LET SYSTEM MANAGER HANDLE MENU KEYSTROKE*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0204h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to menu data (see Table 20-8) |
| | WORD | keystroke |
| | DWORD | pointer to WORD to receive selection number |

**Return Registers:**
buffer for selection number filled with index of selected menu item or FFFFh if no final selection yet
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0200h, 0202h, and 0207h

### INT 60h Function 0205h
### *INITIALIZE FILE SELECTION MENU*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0205h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to file menu structure (see Table 20-9) |
| | DWORD | pointer to edit record (see Table 20-11) |
| | DWORD | pointer to wildcard filespec for initial file list |
| | WORD | *row* |
| | WORD | *column* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0200h, 0206h, and 0208h

**20-9 Format of file menu structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to ASCIZ base directory name |
| 04h | DWORD | pointer to ASCIZ file pattern (wildcard filespec) |
| 08h | DWORD | pointer to file list workspace, at least 1024 bytes (see Table 20-10) |
| 0Ch | WORD | size of file list workspace in bytes |

| | | |
|---|---|---|
| 0Eh | WORD | starting row (-3 is topmost, 0 is first non-"reserved" line) |
| 10h | WORD | starting column |
| 12h | WORD | number of lines |
| 14h | WORD | number of columns |
| 16h | WORD | number of files displayed on each line |
| **---the remaining fields are initialized by the System Manager---** | | |
| 18h | WORD | 0000h if first edit character, else multiline |
| 1Ah | WORD | number of files in file list |
| 1Ch | WORD | max files workspace has room for |
| 1Eh | WORD | file at top of list |
| 20h | WORD | index of file to highlight |
| 22h | WORD | index of file to unhighlight |
| 24h | WORD | current focus (01h FMENU, 02h EDIT) |

**20-10 Format of file list workspace entry:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | file attributes |
| 01h | WORD | file time (see Table 24-!I) |
| 03h | WORD | file date (see Table 24-!J) |
| 05h | DWORD | file size |
| 09h | 13 BYTEs | ASCIZ filename |

**INT 60h Function 0206h** *HP 95LX System Manager (partially doc.)*

***DISPLAY/REDISPLAY FILE SELECTION MENU***

**Registers at call:**

DI = 0206h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to file menu structure (see Table 20-9) |
| | DWORD | pointer to edit record (see Table 20-11) |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 0205h

**INT 60h Function 0207h** *HP 95LX System Manager (partially doc.)*

***LET SYSMGR PROCESS FILE SELECTION MENU KEYSTROKE***

**Registers at call:**

DI = 0207h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to file menu (see Table 20-9) |
| | DWORD | pointer to edit record (see Table 20-11) |
| | WORD | keystroke |

**Return Registers:**

AX = status

| | |
|---|---|
| 0000h | keystroke processed, call INT 60h Function 0206h to refresh menu |
| 0001h | redisplay application area before refreshing menu |
| 0002h | user confirmed selection, filename is in edit record's buffer |
| 0003h | user aborted menu |
| FFFBh | bad filename |
| FFFCh | bad directory |
| FFFDh | bad drive |
| FFFEh | unknown keystroke |

FFFFh keystroke known but invalid in current context
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0205h and 0208h

---

## INT 60h Function 0208h
### *REMOVE FILE SELECTION MENU*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

| | | |
|---|---|---|
| DI = 0208h | | |
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to file menu structure (see Table 20-9) |
| | DWORD | pointer to edit record (see Table 20-11) |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0205h and 0206h, INT 15h Function 4DD4h (Chapter 7)

---

## INT 60h Function 0300h
### *DISPLAY STRING*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

| | | |
|---|---|---|
| DI = 0300h | | |
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | starting row (-3 is topmost, 0 is first user line) |
| | WORD | starting column |
| | DWORD | pointer to string |
| | WORD | length of string |
| | WORD | display style: 0000h normal, 0001h reverse video |
| | WORD | "OSTYLE" |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0F03h and 1005h

---

## INT 60h Function 0301h
### *CLEAR PORTION OF SCREEN*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

| | | |
|---|---|---|
| DI = 0301h | | |
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | starting row (-3 is topmost, 0 is first user line) |
| | WORD | starting column |
| | WORD | number of rows |
| | WORD | number of columns |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0302h and 1005h

---

### INT 60h Function 0302h
### *SCROLL PORTION OF SCREEN*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0302h

| STACK: | | |
|---|---|---|
| | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | *starting row* |
| | WORD | *starting column* |
| | WORD | *height of scroll region* |
| | WORD | *width of scroll region* |
| | WORD | *number of lines to scroll region* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0301h

---

### INT 60h Function 0303h
### *SCREEN SERVICE "M_XCHG"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0303h

| STACK: | | |
|---|---|---|
| | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | DWORD | pointer to *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0304h
### *SCREEN SERVICE "M_CHRATTR"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0304h

| STACK: | | |
|---|---|---|
| | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *unknown* |
| | WORD | *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0305h
### *SCREEN SERVICE "M_CHRRVRT"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0305h

| STACK: | | |
|---|---|---|
| | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | DWORD | pointer to *unknown* |
| | WORD | *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

### INT 60h Function 0307h
### *SCREEN SERVICE "M_CHRINV"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0307h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | WORD | *unknown* |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

### INT 60h Function 0308h
### *SCREEN SERVICE "M_ROWS_COLS"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0308h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

### INT 60h Function 0309h
### *SET SCREEN (VIDEO) MODE*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0309h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | new mode |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 030Ah, INT 5Fh Function 00h

### INT 60h Function 030Ah
### *GET SCREEN (VIDEO) MODE*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 030Ah

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 0309h, INT 5Fh Function 00h

### INT 60h Function 030Bh
### *SET CURSOR POSITION*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 030Bh

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | row (-3 is topmost, 0 is first non-reserved line) |
| | WORD | column |

**Return Registers:**

*unknown*

STACK unchanged

**Details:** The cursor is hidden if the specified position is not on the physical display.
**Conflicts:** see Table 6-3
**See also:** INT 61h and INT 62h below, HP 95LX INT 15h Function 4DD4h (Chapter 7), Video INT 10h Function 02h (Chapter 8)

---

### INT 60h Function 0400h
### *"EDIT_INIT"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0400h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to edit record (see Table 20-11) |
| | DWORD | pointer to string to be edited |
| | WORD | initial length of string being edited |
| | WORD | maximum length of edited string |
| | WORD | row of edit field |
| | WORD | leftmost column of edit field |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**20-11 Format of edit record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | current length of edit buffer |
| 02h | BYTE | flag for special processing on first character |
| 03h | BYTE | flags<br>bit 0: tab handling |
| 04h | WORD | editing in prompt window? |
| 06h | DWORD | pointer to top line of prompt window message |
| 0Ah | WORD | length of top line of prompt |
| 0Ch | DWORD | pointer to second line of prompt window message |
| 10h | WORD | length of second line of prompt |
| 12h | 80 BYTEs | workspace for editing |
| 62h | 2 WORDs | line array needed for multi-line editing |
| 66h | 36 BYTEs | multi-line edit record (see Table 20-12) |
| 8Ah | WORD | displayable columns |

**20-12 Format of multi-line edit record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to user-supplied edit buffer |
| 04h | WORD | length of edit buffer |
| 06h | WORD | current cursor position |
| 08h | WORD | starting row of edit area (-3 is topmost, 0 is first user line) |
| 0Ah | WORD | starting column of edit area |
| 0Ch | WORD | height of edit area |
| 0Eh | WORD | width of edit area |
| 10h | WORD | current top row (-3 is topmost, 0 is first user line) |
| 12h | WORD | number of rows displayable |
| 14h | BYTE | cursor column |

| | | |
|---|---|---|
| 15h | BYTE | 01h if buffer has been modified |
| 16h | BYTE | first displayable column (ticker fields only) |
| 17h | BYTE | 01h if wordwrap enabled, FFh if ticker field |
| 18h | DWORD | pointer to array of line starts (at least one bigger than edit area is high) |
| 1Ch | BYTE | currently marking? |
| 1Dh | BYTE | flag |
| 1Eh | WORD | offset of mark start |
| 20h | WORD | offset of mark end (inclusive) |
| 22h | WORD | displayable columns |

---

## INT 60h Function 0401h

*HP 95LX System Manager (partially doc.)*

### *EDIT ON TOP LINE*

**Registers at call:**

DI = 0401h

| STACK: | | |
|---|---|---|
| | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to edit record (see Table 20-11) |
| | DWORD | pointer to string to edit |
| | WORD | initial length of string being edited |
| | WORD | maximum length of edited string |
| | DWORD | pointer to first line of prompt |
| | WORD | length of first line |
| | DWORD | pointer to second line of prompt |
| | WORD | length of second line |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

---

## INT 60h Function 0402h

*HP 95LX System Manager (partially doc.)*

### *DISPLAY OR REDISPLAY EDIT FIELD*

**Registers at call:**

DI = 0402h

| STACK: | | |
|---|---|---|
| | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to edit record (see Table 20-11) |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 0406h

---

## INT 60h Function 0403h

*HP 95LX System Manager (partially doc.)*

### *LET SYSTEM MANAGER PROCESS EDITING KEYSTROKE*

**Registers at call:**

DI = 0403h

| STACK: | | |
|---|---|---|
| | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to edit record (see Table 20-11) |

| | | |
|---|---|---|
| | WORD | keystroke |
| | DWORD | pointer to WORD buffer for result code |

**Return Registers:**
result code buffer filled with 0001h if editing complete
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0406h

---

### INT 60h Function 0404h
### *"MDIT_INI" - INIT MULTI-LINE EDIT FIELD?*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0404h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *unknown* |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | DWORD | pointer to *unknown* |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | DWORD | pointer to *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0405h
### *"MDIT_DIS" - DISPLAY MULTI-LINE EDIT FIELD?*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0405h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *multi-line edit record* (see Table 20-12) |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0402h

---

### INT 60h Function 0406h
### *"MDIT_KEY" - PROCESS MULTI-LINE EDIT KEYSTROKE?*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0406h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *multi-line edit record* (see Table 20-12) |
| | WORD | *keystroke* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0403h

---

### INT 60h Function 0407h
### *"MDIT_FIL"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0407h

STACK: 2 WORDs unused dummies (for calls from high level languages)

DWORD pointer to *unknown*

DWORD pointer to *unknown*

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0408h
### *"MDIT_MARK"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0408h

STACK: 2 WORDs unused dummies (for calls from high level languages)

DWORD pointer to *multi-line edit record* (see Table 20-12)

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Functions 0409h and 040Ah

---

### INT 60h Function 0409h
### *"MDIT_UNMARK"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0409h

STACK: 2 WORDs unused dummies (for calls from high level languages)

DWORD pointer to *multi-line edit record* (see Table 20-12)

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Functions 0408h and 040Ah

---

### INT 60h Function 040Ah
### *"MDIT_CUTMARK"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 040Ah

STACK: 2 WORDs unused dummies (for calls from high level languages)

DWORD pointer to *multi-line edit record* (see Table 20-12)

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Functions 0408h and 0409h

---

### INT 60h Function 040Bh
### *"MDIT_INS_STR" - INSERT STRING IN EDIT FIELD?*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 040Bh

STACK: 2 WORDs unused dummies

(for calls from high level languages)
DWORD pointer to *multi-line edit record* (see Table 20-12)
DWORD pointer to *string to be inserted*
WORD *length of string*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** INT 61h and INT 62h below, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

### INT 60h Function 0500h
### *OPEN FILE*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0500h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to file state record (see Table 20-13)
DWORD pointer to filename
WORD length of filename
WORD *unknown*
WORD suppress buffering if nonzero

**Return Registers:**
AX = status
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0501h, 0502h, and 0508h

**20-13 Format of file state record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | DOS file handle |
| 02h | WORD | flags<br>bit 0: buffer contents valid<br>bit 1: buffer is dirty and must be written<br>bit 2: unbuffered I/O<br>bit 3: file is a character device |
| 04h | DWORD | current DOS physical file offset (FFFFFFFFh if unknown) |
| 08h | DWORD | DOS file offset of start of buffer |
| 0Ch | DWORD | effective file offset as seen by caller |
| 10h | WORD | number of bytes in file buffer |
| **---buffered I/O only---** | | |
| 12h | 512 BYTEs | file buffer |

---

### INT 60h Function 0501h
### *OPEN FILE IN READ-ONLY MODE*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0501h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to file state record (see Table 20-13)
DWORD pointer to filename
WORD length of filename
WORD *unknown*

WORD suppress buffering if nonzero

**Return Registers:**
AX = status
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0500h, MS-DOS INT 21h Function 3Dh (Chapter 24)

---

### INT 60h Function 0502h
### *CREATE NEW FILE*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0502h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to file state record (see Table 20-13)
DWORD pointer to filename
WORD length of filename
WORD *unknown*
WORD suppress buffering if nonzero

**Return Registers:**
AX = status
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0500h and 0503h, MS-DOS INT 21h Function 3Ch (Chapter 24)

---

### INT 60h Function 0503h
### *CREATE OR TRUNCATE FILE*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0503h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to file state record (see Table 20-13)
DWORD pointer to filename
WORD length of filename
WORD *unknown*
WORD suppress buffering if nonzero

**Return Registers:**
AX = status
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0502h, MS-DOS INT 21h Function 3Ch (Chapter 24)

---

### INT 60h Function 0504h
### *READ FROM FILE*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0504h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to file state record (see Table 20-13)
DWORD pointer to data buffer
WORD number of bytes to read
DWORD pointer to WORD in which to return actual bytes read

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0505h, MS-DOS INT 21h Function 3Fh (Chapter 24)

### INT 60h Function 0505h
### *WRITE TO FILE*

*HP 95LX System Manager*

**Registers at call:**

DI = 0505h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to file state record (see Table 20-13) |
| | DWORD | pointer to data |
| | WORD | length of data |

**Return Registers:**

AX = status

STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 0504h, MS-DOS INT 21h Function 40h (Chapter 24)

---

### INT 60h Function 0506h
### *SET FILE POSITION*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0506h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to file state record (see Table 20-13) |
| | WORD | *unknown* |
| | WORD | *unknown* |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 0507h, MS-DOS INT 21h Function 42h (Chapter 24)

---

### INT 60h Function 0507h
### *"M_TELL" - GET FILE POSITION*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0507h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to file state record (see Table 20-13) |
| | DWORD | *pointer to DWORD buffer for file position* |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 0506h, MS-DOS INT 21h Function 42h (Chapter 24)

---

### INT 60h Function 0508h
### *CLOSE FILE*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0508h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to file state record (see Table 20-13) |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 0500h, MS-DOS INT 21h Function 3Eh (Chapter 24)

### INT 60h Function 0509h
### *FILE SERVICE "M_SETPAT"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0509h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *unknown* |
| | DWORD | pointer to *unknown* |
| | WORD | *unknown* |
| | WORD | *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

### INT 60h Function 050Ah
### *FILE SERVICE "M_MATCH"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 050Ah

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *unknown* |
| | DWORD | pointer to *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** MS-DOS INT 21h Function 4Eh (Chapter 24)

**20-14 Format of pattern match control block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 43 BYTEs | FindFirst data block (see Table 24-!K) |
| 2Bh | 80 BYTEs | full path name |
| 7Bh | BYTE | offset of last component of filename |
| 7Ch | BYTE | DOS function number (4Eh or 4Fh) |

### INT 60h Function 050Bh
### *IDENTIFY FILENAME REFERENT*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 050Bh

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *filename* |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | DWORD | pointer to *buffer for file type* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**20-15 File types returned:**

| | |
|---|---|
| 0000h | nonexistent |
| 0001h | file |
| 0002h | directory |
| 0003h | character device |

### INT 60h Function 050Ch
### *"M_DELETE" - DELETE FILE*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 050Ch "M_DELETE"

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *filename or file state record* |
| | WORD | *unknown* |
| | WORD | *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** MS-DOS INT 21h Function 41h (Chapter 24)

### INT 60h Function 050Dh
### *RENAME FILE*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 050Dh

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *old name* |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | DWORD | pointer to *new name* |
| | WORD | *unknown* |
| | WORD | *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** MS-DOS INT 21h Function 56h (Chapter 24)

### INT 60h Function 050Eh
### *FILE SERVICE "M_GETDIR"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 050Eh

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | *unknown* |
| | DWORD | pointer to *unknown* |
| | DWORD | pointer to *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** MS-DOS INT 21h Function 47h (Chapter 24)

### INT 60h Function 050Fh
### *FILE SERVICE "M_SETDIR"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 050Fh

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *directory name* |
| | WORD | *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** MS-DOS INT 21h Function 3Bh (Chapter 24)

---

### INT 60h Function 0510h
### *FILE SERVICE "M_VOLUME"*

*HP 95LX System Manager*
*(partially doc.)*

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| DI = 0510h | | | *unknown* |
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) | STACK unchanged |
| | DWORD | pointer to *unknown* | |
| | DWORD | pointer to *unknown* | |

**Conflicts:** see Table 6-3

---

### INT 60h Function 0511h
### *MAKE A SUBDIRECTORY*

*HP 95LX System Manager*
*(partially doc.)*

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| DI = 0511h | | | *unknown* |
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) | STACK unchanged |
| | DWORD | pointer to *directory name* | |
| | WORD | *unknown* | |
| | WORD | *unknown* | |

**Conflicts:** see Table 6-3
**See also:** Function 0512h, MS-DOS INT 21h Function 39h (Chapter 24)

---

### INT 60h Function 0512h
### *REMOVE A SUBDIRECTORY*

*HP 95LX System Manager*
*(partially doc.)*

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| DI = 0512h | | | *unknown* |
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) | STACK unchanged |
| | DWORD | pointer to *directory name* | |
| | WORD | *unknown* | |
| | WORD | *unknown* | |

**Conflicts:** see Table 6-3
**See also:** Function 0511h, MS-DOS INT 21h Function 3Ah (Chapter 24)

---

### INT 60h Function 0513h
### *GET DEFAULT DRIVE*

*HP 95LX System Manager*
*(partially doc.)*

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| DI = 0513h | | | *unknown* |
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) | STACK unchanged |
| | DWORD | pointer to *unknown* buffer for current drive | |

**Conflicts:** see Table 6-3
**See also:** Function 0514h, MS-DOS INT 21h Function 19h (Chapter 24)

---

**INT 60h Function 0514h** *HP 95LX System Manager (partially doc.)*

***SET DEFAULT DRIVE***

**Registers at call:**
DI = 0514h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD new drive

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 0513h, MS-DOS INT 21h Function 0Eh (Chapter 24)

---

**INT 60h Function 0515h** *HP 95LX System Manager (partially doc.)*

***FILE SERVICE "M_FDATE"***

**Registers at call:**
DI = 0515h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *filename or file state record*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**See also:** MS-DOS INT 21h Functions 5700h and 5701h (Chapter 24)

---

**INT 60h Function 0516h** *HP 95LX System Manager (partially doc.)*

***FILE SERVICE "M_GET_SYSDIR"***

**Registers at call:**
DI = 0516h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to buffer for *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

**INT 60h Function 0517h** *HP 95LX System Manager (partially doc.)*

***GET FILE ATTRIBUTES***

**Registers at call:**
DI = 0517h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *filename or file state record*
WORD *unknown*
WORD *unknown*
DWORD pointer to *unknown* buffer for file's attributes

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 0518h, MS-DOS INT 21h Function 4300h (Chapter 24)

### INT 60h Function 0518h
### *SET FILE ATTRIBUTES*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0518h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *filename or file state record* |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | WORD | *new attributes* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0517h, MS-DOS INT 21h Function 4301h (Chapter 24)

### INT 60h Function 0519h
### *FILE SERVICE "M_COMMON_OPEN"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0519h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *unknown* |
| | DWORD | pointer to *unknown* |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | WORD | *unknown* |
| | WORD | *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

### INT 60h Function 051Ah
### *FILE SERVICE "M_COPYDT"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 051Ah

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *unknown* |
| | DWORD | pointer to *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

### INT 60h Function 051Bh
### *FILE SERVICE "M_GETFDT"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 051Bh

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to *filename or file state record* |
| | DWORD | pointer to *buffer for result* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 051Ch

---

### INT 60h Function 051Ch
### *FILE SERVICE "M_PUTFDT"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 051Ch
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *filename or file state record*
WORD *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 051Bh

---

### INT 60h Function 0600h
### *PROCESS INITIALIZING*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0600h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0601h, HP 95LX INT 61h below, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

### INT 60h Function 0601h
### *PROCESS TERMINATION*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0601h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
never
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0600h, MS-DOS INT 21h Function 4Ch (Chapter 24)

---

### INT 60h Function 0602h
### *"M_LOCK" - PREVENT TASK SWITCHES*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0602h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0603h, TopView INT 15h Function 101Bh (Chapter 44), MS Windows INT 2Fh Function 1681h (Chapter 43)

---

### INT 60h Function 0603h
### *"M_UNLOCK" - ALLOW TASK SWITCHES*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0603h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0602h, TopView INT 15h Function 101Ch (Chapter 44), MS Windows INT 2Fh Function 1682h (Chapter 43)

---

### INT 60h Function 0604h
### *"M_SPAWN" - START NEW APPLICATION*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0604h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*
WORD *unknown*
WORD *unknown*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0600h, 0601h, and 0607h; MS-DOS INT 21h Function 4Bh (Chapter 24)

---

### INT 60h Function 0605h
### *"M_APPCOUNT" - GET NUMBER OF RUNNING APPLICATIONS?*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0605h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0603h

---

### INT 60h Function 0606h
### *"M_REBOOT" - REBOOT SYSTEM?*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0606h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** BIOS INT 19h (Chapter 4)

---

### INT 60h Function 0607h
### *"M_SPAWNARG"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0607h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*
WORD *unknown*
DWORD pointer to *unknown*
WORD *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0604h

---

### INT 60h Function 0608h
### *"M_REG_APP_NAME" - REGISTER APPLICATION NAME?*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0608h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *application name*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0609h
### *"M_APP_NAME" - GET APPLICATION NAME?*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0609h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *buffer for application name*

**Return Registers:**
DX:AX -> *unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** INT 61h and INT 62h below, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

### INT 60h Function 0700h
### *OPEN CLIPBOARD*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0700h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0701h and 0702h, WINOLDAP INT 2Fh Function 1701h (Chapter 43)

**20-16 Values for error code:**

| | |
|---|---|
| 0000h | successful |
| FFF8h | transfer request out of bounds |
| FFF9h | no such representation |
| FFFAh | no representation open |
| FFFBh | a representation is already open |
| FFFCh | representation already exists |
| FFFDh | heap allocation failure |
| FFFEh | clipboard not open |
| FFFFh | clipboard access denied |

---

### INT 60h Function 0701h
### *CLOSE CLIPBOARD*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0701h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0700h and 0702h, WINOLDAP INT 2Fh Function 1708h (Chapter 43)

### INT 60h Function 0702h
### *RESET CLIPBOARD*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0702h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0700h, WINOLDAP INT 2Fh Function 1702h (Chapter 43)

---

### INT 60h Function 0704h
### *"M_NEW_REP" - START A NEW REPRESENTATION*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0704h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0705h, 0706h, and 0707h

---

### INT 60h Function 0705h
### *CLIPBOARD SERVICE "M_FINI_REP"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0705h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0704h

---

### INT 60h Function 0706h
### *CLIPBOARD SERVICE "M_REP_NAME"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0706h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *unknown*
DWORD pointer to *unknown*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0704h and 0707h

---

### INT 60h Function 0707h
### *CLIPBOARD SERVICE "M_REP_INDEX"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0707h
STACK: 2 WORDs unused dummies (for calls from high level languages)

DWORD pointer to *unknown*
DWORD pointer to *unknown*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0704h and 0706h

---

### INT 60h Function 0708h
### *WRITE TO CLIPBOARD*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0708h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD *pointer to data to be written*
WORD *length of data*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0700h and 0709h, WINOLDAP INT 2Fh Function 1703h (Chapter 43)

---

### INT 60h Function 0709h
### *READ FROM CLIPBOARD*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0709h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *unknown*
WORD *unknown*
DWORD *pointer to buffer for data*
WORD *length of buffer*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0708h, WINOLDAP INT 2Fh Function 1705h (Chapter 43)

---

### INT 60h Function 0800h
### *BEEP*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0800h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0801h, 0802h, and 0803h; BIOS INT 10h Function 0Eh (Chapter 8)

---

### INT 60h Function 0801h
### *SOUND SERVICE "M_THUD"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0801h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0800h, 0802h, and 0803h

---

### INT 60h Function 0802h
### *MAKE A SOUND PATTERN*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0802h
STACK: 2 WORDs unused dummies
(for calls from
high level languages)
WORD pattern number (00h-06h)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0800h, 0801h, and 0803h; TopView INT 15h Function 1019h (Chapter 44)

---

### INT 60h Function 0803h
### *TURN OFF SOUND*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0803h
STACK: 2 WORDs unused dummies
(for calls from
high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0800h, 0801h, and 0802h

---

### INT 60h Function 0900h
### *ALLOCATE REGULAR MEMORY BLOCK*

*HP 95LX System Manager*

**Registers at call:**
DI = 0900h
STACK: 2 WORDs unused dummies
(for calls from
high level languages)
WORD size of block in bytes

**Return Registers:**
AX -> memory block
STACK unchanged

**Details:** System Manager-compliant applications are always small-model (64K code, 64K data).
**Conflicts:** see Table 6-3
**See also:** Functions 0902h and 0903h, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

### INT 60h Function 0902h
### *FREE REGULAR MEMORY BLOCK*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0902h
STACK: 2 WORDs unused dummies
(for calls from
high level languages)
WORD *offset of memory block*

**Return Registers:**
*unknown*
STACK unchanged

**Details:** System Manager-compliant applications are always small-model (64K code, 64K data).
**Conflicts:** see Table 6-3
**See also:** Functions 0900h and 0904h

---

### INT 60h Function 0903h
### *ALLOCATE LARGE MEMORY BLOCK*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0903h
STACK: 2 WORDs unused dummies
(for calls from

high level languages)
WORD *size of block in bytes*

**Return Registers:**
AX -> *memory block*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0900h and 0904h

---

### INT 60h Function 0904h
### *FREE LARGE MEMORY BLOCK*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0904h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *segment of memory block*

**Return Registers:**
AX -> *unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0902h and 0903h

---

### INT 60h Function 0B00h
### *CLOCK/CALENDAR SERVICE "M_DTINFO"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B00h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0B01h
### *CLOCK/CALENDAR SERVICE "M_GETDTM"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B01h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0B02h

---

### INT 60h Function 0B02h
### *CLOCK/CALENDAR SERVICE "M_SETDTM"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B02h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0B01h

---

### INT 60h Function 0B03h
### *CLOCK/CALENDAR SERVICE "M_XALARM"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B03h

STACK: 2 WORDs unused dummies
(for calls from
high level languages)
WORD *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0B04h

---

### INT 60h Function 0B04h
### *CLOCK/CALENDAR SERVICE "M_ALARM"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0B04h
STACK: 2 WORDs unused dummies
(for calls from
high level languages)
DWORD *pointer to alarm record*
WORD *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0B03h, BIOS INT 1Ah Function 06h (Chapter 4)

**20-17 Format of alarm record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | hour |
| 01h | BYTE | minute |
| 02h | BYTE | second |
| 03h | BYTE | unused padding |
| 04h | WORD | rescheduling interval, in seconds |
| 06h | BYTE | are seconds significant? |
| 07h | BYTE | alarm sound |
| 08h | 40 BYTEs | message displayed when alarm activates |
| 30h | BYTE | task ID of owner |
| 31h | BYTE | application's own use for sub-class |
| 32h | 4 BYTEs | application's own use for private data |

---

### INT 60h Function 0B05h
### *CLOCK/CALENDAR SERVICE "M_START_SW"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0B05h
STACK: 2 WORDs unused dummies
(for calls from
high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0B06h and 0B07h

---

### INT 60h Function 0B06h
### *CLOCK/CALENDAR SERVICE "M_GET_SW"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0B06h
STACK: 2 WORDs unused dummies
(for calls from
high level languages)
DWORD pointer to *unknown*

DWORD pointer to *unknown*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0B05h and 0B07h

---

### INT 60h Function 0B07h
### *CLOCK/CALENDAR SERVICE "M_STOP_SW"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0B07h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0B05h and 0B06h

---

### INT 60h Function 0B08h
### *"M_TELLTIME" - DISPLAY TIMESTAMP*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0B08h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD timestamp format (see Table 20-18)
WORD row (-3 is topmost, 0 is first non-reserved line)
WORD column

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**20-18 Bitfields for timestamp format:**

| | |
|---|---|
| bits 1-0 | timestamp components<br>00 date only<br>01 time only<br>10 date and time<br>11 day and date |
| bit 4 | supply am/pm |
| bit 5 | supply seconds |
| bit 6 | show year |
| bit 7 | four-digit year |

---

### INT 60h Function 0B09h
### *CLOCK/CALENDAR SERVICE "M_GET_SETTINGS"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0B09h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0B0Ah and 0B0Fh

**20-19 Format of system settings:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | country code |
| 02h | WORD | speaker volume (00h-03h or FFh for off) |
| 04h | WORD | contrast level (00h-0Fh) |
| 06h | WORD | week start (00h Sunday, 01h Monday) |
| 08h | WORD | punctuation format |

| code | decimal | arg | thousands |
|---|---|---|---|
| 00h | . | , | , |
| 01h | , | . | . |
| 02h | . | ; | ; |
| 03h | , | ; | . |
| 04h | . | , | space |
| 05h | , | . | space |
| 06h | . | ; | space |
| 07h | , | ; | space |

| *Offset* | *Size* | *Description* |
|---|---|---|
| 0Ah | WORD | two-character language code (only 5355h = "US" byte-swapped) |
| 0Ch | WORD | current date format (see Table 20-20) |
| 0Eh | WORD | current time format (see Table 20-21) |
| 10h | WORD | collating sequence<br>00h numbers first, 01h letters first, 02h ASCII |
| 12h | 80 BYTEs | name of picture file |
| 62h | 30 BYTEs | name |
| 80h | 30 BYTEs | title |
| 9Eh | 28 BYTEs | company name |
| BAh | WORD | number of languages |
| BCh | 6 BYTEs | available languages |
| C2h | 66 BYTEs | language menu |
| 104h | 2 BYTEs | ASCIZ date separator |
| 106h 2 | BYTEs | ASCIZ time separator |
| 108h | BYTE | date order |
| 109h | BYTE | use 24 hour time? |
| 10Ah | 16 BYTEs | currency string |
| 11Ah | WORD | currency string position (00h prefix, 01h suffix) |
| 11Ch | WORD | keyboard (see Table 20-22) |
| 11Eh | WORD | printer baud rate<br>00h 300, 01h 1200, 02h 2400, 03h 4800, 04h 9600, 05h 19200 |
| 120h | WORD | printer driver code<br>00h Epson FX80, 01h HP Laserjet, 02h IBM ProPrinter |
| 122h | WORD | printer interface (00h COM1, 01h COM2, 02h IR, 03h LPT1) |
| 124h | WORD | system manager interrupt (60h by default) |
| 126h | WORD | code page (01h CP850, 02h CP437) |
| 128h | WORD | active exit key |
| 12Ah | WORD | active menu key |
| 12Ch | WORD | active CHAR key toggle |
| 12Eh | 6 BYTEs | alarm |

**20-20 Date formats:**

| | |
|---|---|
| 00h | dd-mmm-yy |
| 01h | dd-mmm |

| | |
|---|---|
| 02h | mmm-yy |
| 03h | mm/dd/yy |
| 04h | dd/mm/yy |
| 05h | dd.mm.yy |
| 06h | yy-mm-dd |
| 07h | mm/dd |
| 08h | dd/mm |
| 09h | dd.mm |
| 0Ah | mm-dd |

**20-21 Time formats:**

| | |
|---|---|
| 00h | HH:MM:SS am/pm |
| 01h | HH:MM am/pm |
| 02h | HH:MM:SS |
| 03h | HH.MM.SS |
| 04h | HH,MM,SS |
| 05h | HHhMMmSSs |
| 06h | HH:MM |
| 07h | HH.MM |
| 08h | HH,MM |
| 09h | HHhMMm |

**20-22 Keyboard types:**

| | |
|---|---|
| 0001h | Belgium |
| 0002h | French Canadian |
| 0004h | Denmark |
| 0008h | Finland |
| 0010h | French |
| 0020h | Finland |
| 0040h | Italy |
| 0080h | Netherlands |
| 0100h | Norway |
| 0200h | Portugal |
| 0400h | Spain |
| 0800h | Sweden |
| 1000h | Swiss French |
| 2000h | Swiss German |
| 4000h | United Kingdom |
| 8000h | USA |

---

## INT 60h Function 0B0Ah
## *CLOCK/CALENDAR SERVICE "M_SET_SETTINGS"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0B0Ah
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0B09h

---

### INT 60h Function 0B0Bh
### *CLOCK/CALENDAR SERVICE "M_START_TIMER"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B0Bh
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0B0Ch and 0B0Dh

---

### INT 60h Function 0B0Ch
### *CLOCK/CALENDAR SERVICE "M_STOP_TIMER"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B0Ch
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0B0Bh and 0B0Dh

---

### INT 60h Function 0B0Dh
### *CLOCK/CALENDAR SERVICE "M_GET_TIMER"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B0Dh
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*
DWORD pointer to *unknown*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0B0Bh and 0B0Ch

---

### INT 60h Function 0B0Eh
### *CLOCK/CALENDAR SERVICE "M_TELL_ANYTIME"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B0Eh
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *unknown*
WORD *unknown*
WORD *unknown*
DWORD pointer to *unknown*
DWORD pointer to *unknown*

**Return Registers:**
DX:AX -> *unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0B0Fh
### *"M_GET_SETTINGS_ADDR" - GET SYSTEM SETTINGS RECORD*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B0Fh

STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
DX:AX -> system settings record (see Table 20-19)
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0B09h

---

## INT 60h Function 0B10h
### *PARSE DATE SPECIFICATION*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B10h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *unknown*
DWORD pointer to *unknown*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0B11h and 0B12h

---

## INT 60h Function 0B11h
### *PARSE TIME SPECIFICATION*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B11h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *unknown*
DWORD pointer to *unknown*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0B10h and 0B13h

---

## INT 60h Function 0B12h
### *SET DATE PARSING RULE*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B12h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD new parsing rule (see Table 20-23)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0B10h and 0B13h

**20-23 Date parsing rules:**

| | |
|---|---|
| 01h | day-month-year |
| 02h | month-day-year |
| 03h | year-month-day |
| 04h | “DMYO” |
| 05h | “MDYO” |

OR with 08h to get any year

---

### INT 60h Function 0B13h
### *SET TIME PARSING RULE*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B13h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD new parsing rule (see Table 20-24)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0B11h and 0B12h

**20-24 Time parsing rules:**

| | |
|---|---|
| 01h | HH:MM:SS (am/pm) |
| 02h | HH:MM:SS (24hr) |
| 03h | HHMM:SS (24hr) |
| 04h | HH:MM:SS.hh (24hr) |
| 05h | HH:MM (am/pm) |
| 06h | HH:MM (24hr) |
| 07h | HHMM (24hr) |

---

### INT 60h Function 0B14h
### *CLOCK/CALENDAR SERVICE "M_POST_TIME"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B14h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0B15h

---

### INT 60h Function 0B15h
### *CLOCK/CALENDAR SERVICE "M_DAY_TRIGGER"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0B15h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0B14h, INT 61h and INT 62h below, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

### INT 60h Function 0C00h
### *OPEN PRINTER*

*HP 95LX System Manager (partially doc.)*

**Purpose:** Request control of the printer.

**Registers at call:**
DI = 0C00h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0C01h, 0C02h, and 0C03h

---

### INT 60h Function 0C01h
### *CLOSE PRINTER*

*HP 95LX System Manager (partially doc.)*

**Purpose:** Relinquish control of printer.

**Registers at call:**
DI = 0C01h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0C00h

---

### INT 60h Function 0C02h
### *WRITE TO PRINTER*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0C02h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to data to be written
WORD length of data

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0C00h, BIOS INT 17h Function 00h (Chapter 15)

---

### INT 60h Function 0C03h
### *INITIALIZE PRINTER*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0C03h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0C00h, BIOS INT 17h Function 01h (Chapter 15)

---

### INT 60h Function 0C04h
### *"M_TRANS_PRINTER"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0C04h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *unknown*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0C05h
### *"M_FALL_PRINTER"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0C05h
STACK: 2 WORDs unused dummies (for calls from high level languages)

WORD *unknown*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** INT 61h and INT 62h below, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

### INT 60h Function 0E00h
### *COMMUNICATIONS SERVICE "M_COMM_INIT"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E00h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0E01h and 0E02h, BIOS INT 14h Function 00h (Chapter 12)

**20-25 Values for error code:**

| | |
|---|---|
| 0000h | successful |
| FFF1h | "E_BUSY" |
| FFF2h | timeout |
| FFF3h | framing error |
| FFF4h | parity error |
| FFF5h | overrun error |
| FFF6h | "E_EMPTY" |
| FFF7h | "E_CONECT" |
| FFF8h | not open |
| FFF9h | out of memory |
| FFFAh | buffer overflow |
| FFFBh | "E_NOFIT" |
| FFFCh | unsupported |
| FFFDh | "E_IVOPR" |
| FFFEh | "E_IVCHN" |
| FFFFh | "E_REOPEN" |

---

### INT 60h Function 0E01h
### *OPEN COMMUNICATIONS CHANNEL*

*HP 95LX System Manager*
*(partially doc.)*

**Purpose:** Request control of communications channel.

**Registers at call:**
DI = 0E01h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to WORD buffer for comm channel handle
WORD communications line number (01h-04h)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0E00h and 0E02h

---

### INT 60h Function 0E02h
### *CLOSE COMMUNICATIONS CHANNEL*

*HP 95LX System Manager*
*(partially doc.)*

**Purpose:** Relinquish control of communications channel.

**Registers at call:**
DI = 0E02h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD comm channel handle

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0E00h and 0E01h

---

### INT 60h Function 0E03h
### *"M_COMM_GETMDM"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E03h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *comm channel handle*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0E04h
### *"M_COMM_ANSWER" - ANSWER INCOMING CALL?*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E04h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *comm channel handle*
WORD *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0E05h
### *"M_COMM_DIAL"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E05h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *comm channel handle*
DWORD pointer to *dial string*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0E06h
### *RESET COMMUNICATIONS CHANNEL*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E06h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD comm channel handle
WORD reset options (see Table 20-26)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**20-26 Bitfields for reset options:**

| | |
|---|---|
| bit 0 | reset line |
| bit 1 | flush transmit buffer |
| bit 2 | flush receive buffer |
| bit 3 | reset modem |
| bit 4 | reset receiver's ^S state |
| bit 5 | reset transmitter's ^S state |

---

### INT 60h Function 0E07h
### *"M_COMM_HANGUP" - ISSUE HANGUP COMMAND*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E07h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *comm channel handle*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0E08h
### *SEND DATA OVER COMM CHANNEL*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E08h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD comm channel handle
DWORD pointer to data to be sent
WORD option flags
bit 0: send partial buffer
bit 1: turn on receiver after sending
DWORD pointer to WORD containing length of data to be sent

**Return Registers:**
length WORD updated to contain *number of bytes actually sent*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0E09h and 0E0Bh, BIOS INT 14h Function 01h (Chapter 12)

---

### INT 60h Function 0E09h
### *QUERY COMM CHANNEL TRANSMIT QUEUE*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E09h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *comm channel handle*
DWORD pointer to WORD *to get transmit buffer size*
DWORD pointer to WORD *to get free bytes in transmit buffer*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0E08h and 0E0Ah

---

### INT 60h Function 0E0Ah
### *QUERY COMM CHANNEL RECEIVE QUEUE*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E0Ah

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | comm channel handle |
| | DWORD | pointer to WORD to get receive buffer size |
| | DWORD | pointer to WORD to get free bytes in receive buffer |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0E09h and 0E0Bh

---

### INT 60h Function 0E0Bh
### *RECEIVE DATA FROM COMM CHANNEL*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E0Bh

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | comm channel handle |
| | DWORD | pointer to data buffer |
| | DWORD | pointer to WORD (input) length of data buffer (output) number of bytes received |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0E08h and 0E0Ah, BIOS INT 14h Function 02h (Chapter 12)

---

### INT 60h Function 0E0Ch
### *"M_COMM_HAZCMD"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E0Ch

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | *comm channel handle* |
| | DWORD | pointer to *unknown* |
| | WORD | *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0E0Dh
### *"M_COMM_COMAND"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E0Dh

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | *comm channel handle* |
| | DWORD | pointer to *unknown* |
| | WORD | *unknown* |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0E0Eh
### *"M_COMM_BREAK"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0E0Eh

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | *comm channel handle* |
| | WORD | *unknown* |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0E0Fh
### *"M_COMM_FRCXON"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0E0Fh

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | *comm channel handle* |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 0E10h

---

### INT 60h Function 0E10h
### *"M_COMM_FRCXOF"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0E10h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | *comm channel handle* |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 0E0Fh

---

### INT 60h Function 0E11h
### *"M_COMM_SETDTR" - DTR CONTROL?*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0E11h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | *comm channel handle* |
| | WORD | *new DTR state* |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0E12h
### *"M_COMM_XMITNG"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**

DI = 0E12h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | WORD | *comm channel handle* |

**Return Registers:**

*unknown*

STACK unchanged

**Conflicts:** see Table 6-3

### INT 60h Function 0E13h
### *"M_COMM_STATUS"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E13h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *comm channel handle*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

### INT 60h Function 0E14h
### *SET COMMUNICATIONS SETTINGS*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E14h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD comm channel handle
DWORD pointer to comm settings (see Table 20-27)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0E15h

**20-27 Format of comm settings:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | dial type ('T' tone, 'P' pulse) |
| 01h | WORD | baud rate divisor (115200/baud_rate) |
| 03h | BYTE | parity (00h none, 08h odd, 18h even, 28h mark, 38h space) |
| 04h | BYTE | stop bits (00h one, 04h two) |
| 05h | BYTE | data bits - 5 |
| 06h | BYTE | software handshake<br>01h none, 02h XOFF/XON, 04h XOFF/any, 08h ENQ/ACK |
| 07h | BYTE | infrared (01h off, 02h on) |
| 08h | BYTE | duplex (01h half, 02h full) |
| 09h | BYTE | echo (01h echo, 02h no echo) |

### INT 60h Function 0E15h
### *GET COMMUNICATIONS SETTINGS*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E15h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD comm channel handle
DWORD pointer to buffer for settings (see Table 20-27)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0E14h

### INT 60h Function 0E16h
### *"M_COMM_CNFGUR"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E16h
STACK: 2 WORDs unused dummies

(for calls from
high level languages)
WORD *comm channel handle*
WORD *unknown*
WORD *unknown*
WORD *unknown*
WORD *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0E17h
### *"M_COMM_QRYERR"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0E17h
STACK: 2 WORDs unused dummies
(for calls from
high level languages)
WORD *comm channel handle*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** INT 61h and INT 62h below, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

### INT 60h Function 0F00h
### *"M_ERRMSG"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0F00h
STACK: 2 WORDs unused dummies
(for calls from
high level languages)
WORD *unknown*
DWORD pointer to *unknown*
WORD *unknown*
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0F01h
### *DRAW STANDARD TITLE BOX*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0F01h
STACK: 2 WORDs unused dummies
(for calls from
high level languages)
DWORD pointer to ASCIZ title string

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0F03h and 0F04h

---

### INT 60h Function 0F02h
### *"SHOWNAME"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0F02h
STACK: 2 WORDs unused dummies
(for calls from
high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0F03h
### *DISPLAY TWO-LINE MESSAGE BOX*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0F03h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to first line of message |
| | WORD | length of first line |
| | DWORD | pointer to second line of message |
| | WORD | length of second line |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0300h, 0F01h, 0F04h, and 0F09h

---

### INT 60h Function 0F04h
### *REMOVE MESSAGE BOX*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0F04h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0F01h, 0F03h and 0F09h

---

### INT 60h Function 0F05h
### *"M_COM_TIMER_ADDR"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0F05h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |

**Return Registers:**
DX:AX -> *unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0F06h and 0F07h

---

### INT 60h Function 0F06h
### *"M_COM_TIMER_COUNT_ADDR"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0F06h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |

**Return Registers:**
DX:AX -> *unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0F05h and 0F07h

---

### INT 60h Function 0F07h
### *"M_SYS_RSRC_ADDR"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0F07h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |

**Return Registers:**
DX:AX -> *unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0F05h and 0F06h

---

### INT 60h Function 0F08h
### *"M_BIOS_OUTSTR"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0F08h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | *unknown* | |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0300h and 1506h

---

### INT 60h Function 0F09h
### *DISPLAY THREE-LINE MESSAGE BOX*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0F09h

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |
| | DWORD | pointer to first line of message |
| | WORD | length of first line |
| | DWORD | pointer to second line of message |
| | WORD | length of second line |
| | DWORD | pointer to third line of message |
| | WORD | length of third line |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0F01h, 0F03h and 0F04h

---

### INT 60h Function 0F0Ah
### *DISABLE MACROS*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0F0Ah

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0F0Bh

---

### INT 60h Function 0F0Bh
### *ENABLE MACROS*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 0F0Bh

| | | |
|---|---|---|
| STACK: | 2 WORDs | unused dummies (for calls from high level languages) |

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 0F0Ah

---

### INT 60h Function 0F0Ch
***"M_DATE_TIME_SEPS"***

*HP 95LX System Manager (partially doc.)*

**Purpose:** Either determine or specify the date and time separator characters.

**Registers at call:**
DI = 0F0Ch
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0F0Dh
***"M_FORM_FT"***

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0F0Dh
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
DX:AX -> *unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 0F0Eh
***"M_RAM_IV_INFO"***

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 0F0Eh
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
DX:AX -> *unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**See also:** INT 61h and INT 62h below, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

### INT 60h Function 1005h
***"M_DIRTY_SYNC" - FORCE SCREEN UPDATE***

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 1005h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Functions 0300h and 0301h, TopView INT 10h Function FFh (Chapter 44)

---

### INT 60h Function 1200h
***RESOURCE SERVICE "MAP_RESOURCE_FILE"***

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 1200h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 1201h

*HP 95LX System Manager (partially doc.)*

***"GET_RESOURCE_PTR"***

**Registers at call:**
DI = 1201h
STACK: 2 WORDs unused dummies (for calls from high level languages)
WORD *unknown*

**Return Registers:**
DX:AX -> *unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 1202h, INT 61h and INT 62h below, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

### INT 60h Function 1202h

*HP 95LX System Manager (partially doc.)*

***"GET_RSRC_TAB_PTR"***

**Registers at call:**
DI = 1202h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
DX:AX -> *unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Functions 1201h and 1203h

---

### INT 60h Function 1203h

*HP 95LX System Manager (partially doc.)*

***"INIT_SYSGMR_RSRCS"***

**Registers at call:**
DI = 1203h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Function 1202h

---

### INT 60h Function 1300h

*HP 95LX System Manager (partially doc.)*

***INITIALIZE HELP SYSTEM***

**Registers at call:**
DI = 1300h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*
DWORD pointer to *unknown*
WORD *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

**See also:** Functions 1301h, 1302h, and 1303h

---

### INT 60h Function 1301h

*HP 95LX System Manager (partially doc.)*

***DISPLAY HELP***

**Registers at call:**
DI = 1301h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 1300h and 1302h

---

### INT 60h Function 1302h
### *"M_HELP_KEY" - PROCESS HELP KEYSTROKE?*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 1302h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*
WORD *keystroke*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 1301h and 1303h

---

### INT 60h Function 1303h
### *"M_HELP_TERM"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 1303h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 1300h, INT 61h and INT 62h below, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

### INT 60h Function 1400h
### *"M_ColInit"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 1400h
STACK: 2 WORDs unused dummies (for calls from high level languages)

**Return Registers:**
AX = *unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 1401h to 1406h

---

### INT 60h Function 1401h
### *"M_ColCpStr"*

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 1401h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*
WORD *unknown*
DWORD pointer to *unknown*
WORD *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 1402h
### *"M_ColLicsStr"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 1402h
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 1403h

---

### INT 60h Function 1403h
### *"M_ColLicsChar"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 1403h
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 1402h

---

### INT 60h Function 1404h
### *"M_ColToLower" - CONVERT TO LOWER CASE?*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 1404h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*
WORD *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 1406h

---

### INT 60h Function 1405h
### *"M_ColCpSearch"*

*HP 95LX System Manager*
*(partially doc.)*

**Registers at call:**
DI = 1405h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*
WORD *unknown*
DWORD pointer to *unknown*
WORD *unknown*
WORD *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

### INT 60h Function 1406h

*HP 95LX System Manager*
*(partially doc.)*

### *"M_ColToUpper" - CONVERT TO UPPER CASE?*

**Registers at call:**
DI = 1406h
STACK: 2 WORDs unused dummies (for calls from high level languages)
DWORD pointer to *unknown*
WORD *unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 1404h

---

### INT 60h Function 1500h

*HP 95LX System Manager*
*(partially doc.)*

### *"GrDispInit" - INITIALIZE GRAPHICS DISPLAY*

**Registers at call:**
DI = 1500h
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 1501h, INT 5Fh Function 00h

---

### INT 60h Function 1501h

*HP 95LX System Manager*
*(partially doc.)*

### *"GrDispClear" - CLEAR GRAPHICS DISPLAY*

**Registers at call:**
DI = 1501h
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 1500h and 1502h

---

### INT 60h Function 1502h

*HP 95LX System Manager*
*(partially doc.)*

### *"GrDispDot"*

**Registers at call:**
DI = 1502h
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 1503h to 1505h

---

### INT 60h Function 1503h

*HP 95LX System Manager*
*(partially doc.)*

### *"GrDispDraw"*

**Registers at call:**
DI = 1503h
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 1502h and 1504h

---

### INT 60h Function 1504h
*HP 95LX System Manager (partially doc.)*

***"GrDispFill"***

**Registers at call:**
DI = 1504h
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 1501h and 1503h

---

### INT 60h Function 1505h
*HP 95LX System Manager (partially doc.)*

***"GrDispRead"***

**Registers at call:**
DI = 1505h
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 1502h

---

### INT 60h Function 1506h
*HP 95LX System Manager (partially doc.)*

***"GrDispString"***

**Registers at call:**
DI = 1506h
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Functions 0300h and 0F08h, INT 5Fh Function 0Fh

---

### INT 60h Function 1507h
*HP 95LX System Manager (partially doc.)*

***"GrDispPan" - PAN DISPLAY?***

**Registers at call:**
DI = 1507h
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 1508h

---

### INT 60h Function 1508h
*HP 95LX System Manager (partially doc.)*

***"GrDispZoom" - ZOOM DISPLAY***

**Registers at call:**
DI = 1508h
STACK: 2 WORDs unused dummies (for calls from

high level languages)

*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 1507h

---

**INT 60h Function 1509h**
***"GrDispSave"***

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 1509h
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 150Ah

---

**INT 60h Function 150Ah**
***"GrDispRestore"***

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 150Ah
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** Function 1509h

---

**INT 60h Function 150Bh**
***"GrDispCorner"***

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 150Bh
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3
**See also:** INT 61h and INT 62h below, HP 95LX INT 15h Function 4DD4h (Chapter 7)

---

**INT 60h Function 1604h**
***"CP_TO_LICS"***

*HP 95LX System Manager (partially doc.)*

**Registers at call:**
DI = 1604h
STACK: 2 WORDs unused dummies (for calls from high level languages)
*unknown*

**Return Registers:**
*unknown*
STACK unchanged

**Conflicts:** see Table 6-3

---

**INT 61h**
***LOAD DS***

*HP 95LX System Manager*

**Conflicts:** see Table 6-3
**See also:** HP 95LX INT 0Fh (Chapter 7), INT 60h functions above

---

### INT 62h
*HP 95LX*

### *USED BY CALCULATOR*

**Conflicts:** see Table 6-3
**See also:** INT 60h functions above

---

### INT 63h Function 00h
*HP 100LX*

### *MAP HIGH MEMORY*

**Registers at call:**
AH = 00h
AL = physical page (00h seg C000, 01h seg C400h, ...)
BX = zero-based logical page
CX = page number
DX = device ID (00h system ROM, 05h plugin, etc.)

**Return Registers:**
*unknown*

**Conflicts:** see Table 6-3
**See also:** Function 01h

---

### INT 63h Function 01h
*HP 100LX*

### *SAVE/RESTORE MEMORY MAP*

**Registers at call:**
AH = 01h
AL = function (00h save, 01h restore)
*others, if any, unknown*

**Return Registers:**
*unknown*

**Conflicts:** see Table 6-3
**See also:** Function 00h

## CHAPTER 21

# *Mouse Support*

The mouse has taken a place almost equal to that of the keyboard as an input device, due largely to the rise in popularity of graphical user interfaces. The ability to use nearly any mouse with any mouse-aware application is the result of the *de facto* standard set by Microsoft's mouse driver, which virtually all mice and many other pointing devices support. This chapter describes the Microsoft Mouse INT 33h interface as well as extensions provided by non-Microsoft mouse drivers such as Logitech and Mouse Systems.

## PS/2 Pointing Device

In IBM's PS/2 line, the mouse interface is implemented using techniques that differ from those used in other MS-DOS systems. This section describes the API that is specific to PS/2 systems.

---

### INT 15h Function C200h

*POINTING DEVICE BIOS INTERFACE*

***ENABLE/DISABLE***

*PS/2*

**Purpose:** This function enables or disables the pointing device.

**Registers at call:**
AX = C200h
BH = new state
(00h disabled, 01h enabled)

**Return Registers:**
CF set on error
AH = status (see Table 21-1)

**See also:** Functions C201h and C205h

**21-1 Values for BIOS function status:**

| | |
|---|---|
| 00h | successful |
| 01h | invalid function |
| 02h | invalid input |
| 03h | interface error |
| 04h | need to resend |
| 05h | no device handler installed |

---

### INT 15h Function C201h

*POINTING DEVICE BIOS INTERFACE*

***RESET***

*PS/2*

**Purpose:** This function resets the pointing device.

**Registers at call:**
AX = C201h

**Return Registers:**
CF set on error
    AH = status (see Table 21-1)
CF clear if successful
    BH = device ID
    BL = value returned by attached device after reset
        AAh if device is a mouse

**Details:** After successful completion of this call, the pointing device is set as follows: disabled, sample rate 100 Hz, resolution 4 counts/mm, scaling 1:1, unchanged data package size.
**See also:** Function C205h, INT 33h Function 0000h (below)

---

### INT 15h Function C202h

***SET SAMPLING RATE***

*POINTING DEVICE BIOS INTERFACE*
*PS/2*

**Purpose:** This function specifies how often the pointing device should report its position and button states.

**Registers at call:**
AX = C202h
BH = sampling rate (see Table 21-2)

**Return Registers:**
CF set on error
AH = status (see Table 21-1)

**See also:** Functions C203h and C205h, INT 33h Function 001Ch (below)

**21-2 Values for sampling rate:**

| | |
|---|---|
| 00h | 10/second |
| 01h | 20/second |
| 02h | 40/second |
| 03h | 60/second |
| 04h | 80/second |
| 05h | 100/second |
| 06h | 200/second |

---

### INT 15h Function C203h

***SET RESOLUTION***

*POINTING DEVICE BIOS INTERFACE*
*PS/2*

**Purpose:** Establishes resolution for pointing device.

**Registers at call:**
AX = C203h
BH = resolution (see Table 21-3)

**Return Registers:**
CF set on error
AH = status (see Table 21-1)

**See also:** Functions C202h and C205h

**21-3 Values for resolution:**

| | |
|---|---|
| 00h | one count per mm (25 dpi) |
| 01h | two counts per mm |
| 02h | four counts per mm |
| 03h | eight counts per mm (200 dpi) |

---

### INT 15h Function C204h

***GET TYPE***

*POINTING DEVICE BIOS INTERFACE*
*PS/2*

**Purpose:** Obtains device ID for pointing device.

**Registers at call:**
AX = C204h

**Return Registers:**
CF set on error
AH = status (see Table 21-1)
CF clear if successful
BH = device ID

---

### INT 15h Function C205h

***INITIALIZE***

*POINTING DEVICE BIOS INTERFACE*
*PS/2*

**Purpose:** Initializes the pointing device.

**Registers at call:**
AX = C205h
BH = data package size (1 - 8 bytes)

**Return Registers:**
CF set on error
AH = status (see Table 21-1)

**Details:** The pointing device is set as follows: disabled, 100 Hz sample rate, resolution 4 counts/mm, scaling 1:1.
**See also:** Functions C201h to C203h

### INT 15h Function C206h
*POINTING DEVICE BIOS INTERFACE*

### *EXTENDED COMMANDS*
*PS/2*

**Purpose:** Provides added commands for the pointing device.

**Registers at call:**
AX = C206h
BH = subfunction
    00h return device status
    01h set scaling at 1:1
    02h set scaling at 2:1

**Return Registers:**
CF set on error
    AH = status (see Table 21-1)
CF clear if successful
    **for subfunction 00h:**
    BL = pointing device status (see Table 21-4)
    CL = resolution (see Table 21-3)
    DL = sample rate, reports per second

**21-4 Bitfields for pointing device status:**

| | |
|---|---|
| bit 0 | right button pressed |
| bit 1 | reserved |
| bit 2 | left button pressed |
| bit 3 | reserved |
| bit 4 | 0 if 1:1 scaling, 1 if 2:1 scaling |
| bit 5 | device enabled |
| bit 6 | 0 if stream mode, 1 if remote mode |
| bit 7 | reserved |

### INT 15h Function C207h
*POINTING DEVICE BIOS INTERFACE*

### *SET DEVICE HANDLER ADDRESS*
*PS/2*

**Purpose:** Specify the address of a user handler for the pointing device.

**Registers at call:**
AX = C207h
ES:BX -> FAR user device handler

**Return Registers:**
CF set on error
    AH = status (see Table 21-1)

**Details:** When the subroutine is called, it is passed the following values on the stack; the handler should return with a FAR return without popping the stack:

    WORD 1: status (see table 21-5)
    WORD 2: X data (high byte = 00h)
    WORD 3: Y data (high byte = 00h)
    WORD 4: 0000h.

**See also:** Function C205h, INT 33h Function 000Ch (below)

**21-5 Bitfields for status:**

| | |
|---|---|
| bits 15-8 | reserved (0) |
| bit 7 | Y data overflowed |
| bit 6 | X data overflowed |
| bit 5 | Y data is negative |
| bit 4 | X data is negative |
| bit 3 | reserved (1) |
| bit 2 | reserved (0) |
| bit 1 | right button pressed |
| bit 0 | left button pressed |

## Microsoft Mouse

The Microsoft mouse driver is the acknowledged standard of the MS-DOS world, and is used by nearly all programs as the interface to the rodent. Note that versions prior to 6.24 may not fully support video displays having more than 25 lines of text, or high-resolution video graphics modes.

Neither do all "Microsoft-compatible" drivers from other vendors necessarily support all the functions provided by the Microsoft driver of the same version they report through Function 0024h.

In addition to the INT 33h interface to mouse functionality proper, most mouse drivers also support the EGA Register Interface Library, which is covered in Chapter 8.

---

### INT 33h Function 0000h
### *RESET DRIVER AND READ STATUS*

*MS MOUSE*

**Purpose:** Resets the mouse driver.

**Registers at call:**
AX = 0000h

**Return Registers:**
AX = status
    0000h hardware/driver not installed
    FFFFh hardware/driver installed
BX = number of buttons
    0000h other than two
    0002h two buttons (many drivers)
    0003h Mouse Systems/Logitech three-button mouse
    FFFFh two buttons

**Details:** To use the mouse on a Hercules-compatible monographics card in graphics mode, you must first set 0040h:0049h to 6 for page 0 or 5 for page 1, and then call this function. Logitech drivers v5.01 and v6.00 reportedly do not correctly use Hercules graphics in dual-monitor systems, while version 4.10 does.

The Logitech mouse driver contains the signature string "LOGITECH" three bytes past the interrupt handler; many of the Logitech mouse utilities check for this signature.

**See also:** Functions 0011h, 0021h, and 002Fh; INT 15h Function C201h; FastGraph/Light INT 62h Function 007Ah (Chapter 52), IRQ12 INT 74h (Chapter 3)

---

### INT 33h Function 0001h
### *SHOW MOUSE CURSOR*

*MS MOUSE*

**Registers at call:**
AX = 0001h

**Return Registers:**
none

**Details:** If Function 0002h was called multiple times, multiple calls to this function will be required to actually make the mouse pointer visible. This behavior allows multiple programs to temporarily hide the mouse cursor without requiring them to be aware of other programs that may have hidden the pointer already.

**See also:** Function 0002h, PC Tools INT 16h Function FFFEh (Chapter 61), FastGraph/Light INT 62h Function 007Bh (Chapter 52)

---

### INT 33h Function 0002h
### *HIDE MOUSE CURSOR*

*MS MOUSE*

**Registers at call:**
AX = 0002h

**Return Registers:**
none

**Details:** Multiple calls to hide the cursor will require multiple calls to Function 0001h to unhide it.

**See also:** Functions 0001h and 0010h, PC Tools INT 16h Function FFFFh (Chapter 61), FastGraph/Light INT 62h Function 007Bh (Chapter 52)

---

### INT 33h Function 0003h
### *RETURN POSITION AND BUTTON STATUS*

*MS MOUSE*

**Registers at call:**
AX = 0003h

**Return Registers:**
BX = button status (see Table 21-6)
CX = column
DX = row

**See also:** Functions 0004h and 000Bh, ZWmous INT 2Fh Function D000h (below)

**21-6 Bitfields for button status:**

| | |
|---|---|
| bit 0 | left button pressed if 1 |
| bit 1 | right button pressed if 1 |
| bit 2 | middle button pressed if 1 (Mouse Systems/Logitech mouse) |

### INT 33h Function 0004h

*MS MOUSE*

#### *POSITION MOUSE CURSOR*

**Registers at call:**
AX = 0004h
CX = column
DX = row

**Return Registers:**
none

**Details:** The row and column are truncated to the next lower multiple of the cell size; however, some versions of the Microsoft documentation incorrectly state that the coordinates are rounded.

**See also:** Function 0003h, FastGraph/Light INT 62h Function 0081h (Chapter 52)

### INT 33h Function 0005h

*MS MOUSE*

#### *RETURN BUTTON PRESS DATA*

**Registers at call:**
AX = 0005h
BX = button
  0000h left
  0001h right
  0002h middle (Mouse Systems/Logitech mouse)

**Return Registers:**
AX = button states (see Table 21-6)
BX = number of times specified button has been pressed since last call
CX = column at time specified button was last pressed
DX = row at time specified button was last pressed

**See also:** Function 0006h, FastGraph/Light INT 62h Function 007Ch (Chapter 52)

### INT 33h Function 0006h

*MS MOUSE*

#### *RETURN BUTTON RELEASE DATA*

**Registers at call:**
AX = 0006h
BX = button
  0000h left
  0001h right
  0002h middle (Mouse Systems/Logitech mouse)

**Return Registers:**
AX = button states (see Table 21-6)
BX = number of times specified button has been released since last call
CX = column at time specified button was last released
DX = row at time specified button was last released

**See also:** Function 0005h, FastGraph/Light INT 62h Function 007Ch (Chapter 52)

### INT 33h Function 0007h

*MS MOUSE*

#### *DEFINE HORIZONTAL CURSOR RANGE*

**Purpose:** Limit the horizontal motion of the mouse pointer to a portion of the screen.

**Registers at call:**
AX = 0007h
CX = minimum column
DX = maximum column

**Return Registers:**
none

**See also:** Functions 0008h, 0010h, and 0031h; FastGraph/Light INT 62h Function 0080h (Chapter 52)

### INT 33h Function 0008h

*MS MOUSE*

#### *DEFINE VERTICAL CURSOR RANGE*

**Purpose:** Limit the vertical motion of the mouse pointer to a portion of the screen.

**Registers at call:**
AX = 0008h
CX = minimum row
DX = maximum row

**Return Registers:**
none

**See also:** Functions 0007h, 0010h, and 0031h; FastGraph/Light INT 62h Function 0080h (Chapter 52)

---

### INT 33h Function 0009h

*MS MOUSE*

### *DEFINE GRAPHICS CURSOR*

**Registers at call:**
AX = 0009h
BX = column of cursor hot spot in bitmap (-16 to 16)
CX = row of cursor hot spot (-16 to 16)
ES:DX -> cursor bitmap (see Table 21-7)

**Return Registers:**
none

**See also:** Functions 000Ah, 0012h, and 002Ah; FastGraph/Light INT 62h Function 007Fh (Chapter 52)

**21-7 Format of mouse cursor bitmap:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 16 WORDs | screen mask |
| 20h | 16 WORDs | cursor mask |

**Note:** each word defines the sixteen pixels of a row, low bit rightmost.

---

### INT 33h Function 000Ah

*MS MOUSE*

### *DEFINE TEXT CURSOR*

**Registers at call:**
AX = 000Ah
BX = hardware/software text cursor
  0000h software
    CX = screen mask
    DX = cursor mask
  0001h hardware
    CX = start scan line
    DX = end scan line

**Return Registers:**
none

**Details:** When the software cursor is selected, the character/attribute word at the current screen position is ANDed with the screen mask and then XORed with the cursor mask. CL will typically be set to FFh and DL to 00h in order to leave the character unaltered and only modify its colors.

**See also:** Function 0009h, FastGraph/Light INT 62h Function 007Eh (Chapter 52)

---

### INT 33h Function 000Bh

*MS MOUSE*

### *READ MOTION COUNTERS*

**Registers at call:**
AX = 000Bh

**Return Registers:**
CX = number of mickeys mouse moved horizontally since last call
DX = number of mickeys mouse moved vertically

**Details:** A mickey is the smallest increment the mouse can sense. Positive values indicate down/right.

**See also:** Functions 0003h, 001Bh, and 0027h, Amstrad INT 15h Function 00h (below)

---

### INT 33h Function 000Ch

*MS MOUSE*

### *DEFINE INTERRUPT SUBROUTINE PARAMETERS*

**Registers at call:**
AX = 000Ch
CX = call mask (see Table 21-8)
ES:DX -> FAR routine

**Return Registers:**
none

**Details:** When the subroutine is called, it is passed the following values:

AX = condition mask (same bit assignments as call mask)
BX = button state
CX = cursor column
DX = cursor row
SI = horizontal mickey count
DI = vertical mickey count.

Some versions of the Microsoft documentation incorrectly state that CX bit 0 means call if mouse cursor moves, and swap the meanings of SI and DI.

**See also:** Function 0018h

#### 21-8 Bitfields for call mask:

| | |
|---|---|
| bit 0 | call if mouse moves |
| bit 1 | call if left button pressed |
| bit 2 | call if left button released |
| bit 3 | call if right button pressed |
| bit 4 | call if right button released |
| bit 5 | call if middle button pressed (Mouse Systems/Logitech mouse) |
| bit 6 | call if middle button released (Mouse Sys/Logitech mouse) |

### INT 33h Function 000Dh

*MS MOUSE*

#### *LIGHT PEN EMULATION ON*

| Registers at call: | Return Registers: |
|---|---|
| AX = 000Dh | none |

**See also:** Function 000Eh, INT 10h Function 04h (Chapter 8)

### INT 33h Function 000Eh

*MS MOUSE*

#### *LIGHT PEN EMULATION OFF*

| Registers at call: | Return Registers: |
|---|---|
| AX = 000Eh | none |

**See also:** Function 000Dh, INT 10h Function 04h (Chapter 8)

### INT 33h Function 000Fh

*MS MOUSE*

#### *DEFINE MICKEY/PIXEL RATIO*

| Registers at call: | Return Registers: |
|---|---|
| AX = 000Fh<br>CX = number of mickeys per 8 pixels horizontally (default 8)<br>DX = number of mickeys per 8 pixels vertically (default 16) | none |

**See also:** Functions 0013h and 001Ah, FastGraph/Light INT 62h Function 0082h (Chapter 52)

### INT 33h Function 0010h

*MS MOUSE*

#### *DEFINE SCREEN REGION FOR UPDATING*

**Purpose:** Ensure that the mouse pointer is not displayed on a section of the screen which is being modified by the application.

| Registers at call: | Return Registers: |
|---|---|
| AX = 0010h<br>CX,DX = X,Y coordinates of upper left corner<br>SI,DI = X,Y coordinates of lower right corner | none |

**Details:** The mouse cursor is hidden in the specified region, and needs to be explicitly turned on again.

**See also:** Functions 0001h, 0002h, 0007h, and 0031h; Genius MOUSE Function 0010h

---

**INT 33h Function 0012h** *MS MOUSE*

***SET LARGE GRAPHICS CURSOR BLOCK***

**Registers at call:**
AX = 0012h
BH = cursor width in words
CH = rows in cursor
BL = horizontal hot spot (-16 to 16)
CL = vertical hot spot (-16 to 16)
ES:DX -> bitmap of screen and cursor maps

**Return Registers:**
AX = FFFFh if successful

**See also:** Functions 0009h, 002Ah, and 0035h

---

**INT 33h Function 0013h** *MS MOUSE*

***DEFINE DOUBLE-SPEED THRESHOLD***

**Registers at call:**
AX = 0013h
DX = threshold speed in mickeys/second,
  0000h = default of 64/second

**Return Registers:**
none

**Details:** If the mouse's speed exceeds the threshold, the cursor's on-screen motion is doubled.

**See also:** Functions 000Fh, 001Bh, and 002Ch

---

**INT 33h Function 0014h** *MS MOUSE*

***EXCHANGE INTERRUPT SUBROUTINES***

**Purpose:** Replace the current user mouse event handler with a new one, returning the current handler's address to allow it to be restored later.

**Registers at call:**
AX = 0014h
CX = call mask (see Table 21-8)
ES:DX -> FAR routine

**Return Registers:**
CX = call mask of previous interrupt routine
ES:DX = FAR address of previous interrupt routine

**See also:** Functions 000Ch and 0018h

---

**INT 33h Function 0015h** *MS MOUSE*

***RETURN DRIVER STORAGE REQUIREMENTS***

**Registers at call:**
AX = 0015h

**Return Registers:**
BX = size of buffer needed to store driver state

**See also:** Functions 0016h and 0017h, Mouse Systems Function 0042h (below)

---

**INT 33h Function 0016h** *MS MOUSE*

***SAVE DRIVER STATE***

**Registers at call:**
AX = 0016h
BX = size of buffer (see Function 0015h)
ES:DX -> buffer for driver state

**Return Registers:**
none

**Details:** Although not documented, many drivers appear to require BX on input.

**See also:** Functions 0015h and 0017h

---

**INT 33h Function 0017h** *MS MOUSE*

***RESTORE DRIVER STATE***

**Registers at call:**
AX = 0017h
BX = size of buffer (see Function 0015h)
ES:DX -> buffer containing saved state

**Return Registers:**
none

**Details:** Although not documented, many drivers appear to require BX on input.

Some mouse drivers range-check the values in the saved state based on the current video mode; thus, the video mode should be restored before the mouse driver's state is restored.
**See also:** Functions 0015h and 0016h

---

### INT 33h Function 0018h
*MS MOUSE*
### *SET ALTERNATE MOUSE USER HANDLER*

| Registers at call: | Return Registers: |
|---|---|
| AX = 0018h | AX = status |
| CX = call mask (see Table 21-9) | 0018h if successful |
| ES:DX -> FAR routine | FFFFh on error |

**Details:** When the subroutine is called, it is passed the following values:

AX = condition mask (same bit assignments as call mask)
BX = button state
CX = cursor column
DX = cursor row
SI = horizontal mickey count
DI = vertical mickey count.

Up to three handlers can be defined by separate calls to this function; a handler may be cancelled by calling this function with its address and CX=0000h.

Some versions of the documentation erroneously reverse the order of the bits in the call mask.
**See also:** Functions 000Ch, 0014h, and 0019h

**21-9 Bitfields for call mask:**

| | |
|---|---|
| bit 0 | call if mouse moves |
| bit 1 | call if left button pressed |
| bit 2 | call if left button released |
| bit 3 | call if right button pressed |
| bit 4 | call if right button released |
| bit 5 | call if shift button pressed during event |
| bit 6 | call if ctrl key pressed during event |
| bit 7 | call if alt key pressed during event |

---

### INT 33h Function 0019h
*MS MOUSE*
### *RETURN USER ALTERNATE INTERRUPT VECTOR*

| Registers at call: | Return Registers: |
|---|---|
| AX = 0019h | BX:DX = user interrupt vector |
| CX = call mask (see Table 21-9) | CX = call mask (0000h if not found) |

**Details:** Attempts to find a user event handler (defined by function 18h) whose call mask matches CX.
**See also:** Function 0018h

---

### INT 33h Function 001Ah
*MS MOUSE*
### *SET MOUSE SENSITIVITY*

| Registers at call: | Return Registers: |
|---|---|
| AX = 001Ah | none |
| BX = horizontal speed (see Function 000Fh) | |
| CX = vertical speed (see Function 000Fh) | |
| DX = double speed threshold (see Function 0013h) | |

**See also:** Functions 000Fh, 0013h and 001Bh; FastGraph/Light INT 62h Function 0082h (Chapter 52)

### INT 33h Function 001Bh

*MS MOUSE*

### *RETURN MOUSE SENSITIVITY*

**Registers at call:**
AX = 001Bh

**Return Registers:**
BX = horizontal speed (see Function 000Fh)
CX = vertical speed (see Function 000Fh)
DX = double speed threshold (see Function 0013h)

**See also:** Functions 000Bh and 001Ah

### INT 33h Function 001Ch

*MS MOUSE*
*InPort mouse only*

### *SET INTERRUPT RATE*

**Registers at call:**
AX = 001Ch
BX = rate
  00h no interrupts allowed
  01h 30 per second
  02h 50 per second
  03h 100 per second
  04h 200 per second

**Return Registers:**
none

**Details:** Values greater than 4 may cause unpredictable driver behavior.

### INT 33h Function 001Dh

*MS MOUSE*

### *DEFINE DISPLAY PAGE NUMBER*

**Purpose:** Specify on which display page the mouse cursor will be displayed.

**Registers at call:**
AX = 001Dh
BX = display page number

**Return Registers:**
none

**See also:** Function 001Eh

### INT 33h Function 001Eh

*MS MOUSE*

### *RETURN DISPLAY PAGE NUMBER*

**Purpose:** Determine on which display page the mouse cursor is being displayed.

**Registers at call:**
AX = 001Eh

**Return Registers:**
BX = display page number

**See also:** Function 001Dh

### INT 33h Function 001Fh

*MS MOUSE*

### *DISABLE MOUSE DRIVER*

**Registers at call:**
AX = 001Fh

**Return Registers:**
AX = status
  001Fh unsuccessful
  FFFFh successful
ES:BX = vector for INT 33h before mouse driver was first installed

**Details:** This function restores the vectors for INT 10h and INT 71h (8086) or INT 74h (286/386). If you restore INT 33h to ES:BX, the driver will be completely disabled.

Many drivers return AX=001Fh even though the driver has been disabled.

**See also:** Function 0020h

### INT 33h Function 0020h

*MS MOUSE*

### *ENABLE MOUSE DRIVER*

**Registers at call:**
AX = 0020h

**Return Registers:**
AX = status
  0020h unsuccessful (or driver not installed)
  FFFFh successful

### INT 33h Function 001Bh

**Details:** This function re-hooks the vectors for INT 10h and INT 71h (8086) or INT 74h (286/386) which were removed by Function 001Fh.

Many drivers return AX=0020h even though the driver has been enabled.

**See also:** Function 001Fh

---

### INT 33h Function 0021h
*MS MOUSE*

### *SOFTWARE RESET*

**Registers at call:**
AX = 0021h

**Return Registers:**
AX = status
FFFFh if mouse driver installed
BX = number of buttons (FFFFh = two buttons
0021h if mouse driver not installed

**Details:** This function is identical to Funtion 0000h, but does not reset the mouse.

**See also:** Function 0000h

---

### INT 33h Function 0022h
*MS MOUSE*

### *SET LANGUAGE FOR MESSAGES*

**Registers at call:**
AX = 0022h
BX = language (see Table 21-10)

**Return Registers:**
none

**Details:** This function is only available on international versions of the driver; US versions ignore this call.

**See also:** Function 0023h

**21-10 Values for mouse messages language:**

| | |
|---|---|
| 00h | English |
| 01h | French |
| 02h | Dutch |
| 03h | German |
| 04h | Swedish |
| 05h | Finnish |
| 06h | Spanish |
| 07h | Portugese |
| 08h | Italian |

---

### INT 33h Function 0023h
*MS MOUSE*

### *GET LANGUAGE FOR MESSAGES*

**Registers at call:**
AX = 0023h

**Return Registers:**
BX = language (see Table 21-10)

**Details:** The US version of the driver always returns zero (English).

**See also:** Function 0022h

---

### INT 33h Function 0024h
*MS MOUSE v6.26+*

### *GET SOFTWARE VERSION, MOUSE TYPE, AND IRQ NUMBER*

**Registers at call:**
AX = 0024h

**Return Registers:**
AX = FFFFh on error
otherwise,
BH = major version
BL = minor version
CH = type (1=bus, 2=serial, 3=InPort, 4=PS/2, 5=HP)
CL = interrupt (0=PS/2, 2=IRQ2, 3=IRQ3,...,7=IRQ7)

**See also:** Functions 004Dh and 006Dh

### INT 33h Function 0025h
*MS MOUSE v6.26+*

### *GET GENERAL DRIVER INFORMATION*

**Registers at call:**
AX = 0025h

**Return Registers:**
AX = general information (see Table 21-11)
BX = cursor lock flag for OS/2 to prevent reentrancy problems
CX = mouse code active flag (for OS/2)
DX = mouse driver busy flag (for OS/2)

**21-11 Bitfields for general information:**

| | |
|---|---|
| bit 15 | driver loaded as device driver rather than TSR |
| bit 14 | driver is newer integrated type |
| bits 13,12 | current cursor type |
| | 00 software text cursor |
| | 01 hardware text cursor |
| | 1X graphics cursor |
| bits 11-8 | interrupt rate (see Function 001Ch) |
| bits 7-0 | count of currently-active Mouse Display Drivers (MDD), the newer integrated driver typ |

### INT 33h Function 0026h
*MS MOUSE v6.26+*

### *GET MAXIMUM VIRTUAL COORDINATES*

**Registers at call:**
AX = 0026h

**Return Registers:**
BX = mouse-disabled flag
CX = maximum virtual X (for current video mode)
DX = maximum virtual Y

**See also:** Function 0031h

### INT 33h Function 0027h
*MS MOUSE v7.01+*

### *GET SCREEN/CURSOR MASKS AND MICKEY COUNTS*

**Registers at call:**
AX = 0027h

**Return Registers:**
AX = screen-mask value (or hardware cursor scan-line start for v7.02+)
BX = cursor-mask value (or hardware cursor scan-line stop for v7.02+)
CX = horizontal mickeys moved since last call
DX = vertical mickeys moved since last call

**See also:** Function 000Bh

### INT 33h Function 0028h
*MS MOUSE v7.0+*

### *SET VIDEO MODE*

**Registers at call:**
AX = 0028h
CX = new video mode (call is NOP if 0000h)
DH = Y font size (00h = default)
DL = X font size (00h = default)

**Return Registers:**
CL = status (00h = successful)

**Details:** DX is ignored unless the selected video mode supports font size control.
**See also:** Function 0029h, INT 10h Function 00h (Chapter 8)

### INT 33h Function 0029h
*MS MOUSE v7.0+*

### *ENUMERATE VIDEO MODES*

**Registers at call:**
AX = 0029h
CX = previous video mode

0000h get first supported video mode
other get next supported mode after mode CX

**Return Registers:**
CX = first/next video mode
(0000h = no more video modes)
DS:DX -> description of video mode
or 0000h:0000h if none

**Details:** The enumerated video modes may be in any order and may repeat.

The description string (if available) is terminated by '$' followed by a NUL byte.

**See also:** Function 0028h

---

### INT 33h Function 002Ah

*MS MOUSE v7.02+*

### *GET CURSOR HOT SPOT*

**Registers at call:**
AX = 002Ah

**Return Registers:**
AX = internal counter controlling cursor visibility
BX = cursor hot spot column
CX = cursor hot spot row
DX = mouse type (00h none, 01h bus, 02h serial, 03h InPort, 04h IBM, 05h Hewlett-Packard)

**Details:** The hot spot location is relative to the upper left corner of the cursor block and may range from -128 to +127 both horizontally and vertically.

**See also:** Functions 0009h, 0012h, and 0035h

---

### INT 33h Function 002Bh

*MS MOUSE v7.0+*

### *LOAD ACCELERATION PROFILES*

**Registers at call:**
AX = 002Bh
BX = active acceleration profile
0001h-0004h or FFFFh to restore default curves
ES:SI -> buffer containing acceleration profile data (see Table 21-12)

**Return Registers:**
AX = success flag

**See also:** Functions 002Ch, 002Dh, and 0033h

**21-12 Format of acceleration profile data:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | length of acceleration profile 1 |
| 01h | BYTE | length of acceleration profile 2 |
| 02h | BYTE | length of acceleration profile 3 |
| 03h | BYTE | length of acceleration profile 4 |
| 04h | 32 BYTEs | threshold speeds for acceleration profile 1 |
| 24h | 32 BYTEs | threshold speeds for acceleration profile 2 |
| 44h | 32 BYTEs | threshold speeds for acceleration profile 3 |
| 64h | 32 BYTEs | threshold speeds for acceleration profile 4 |
| 84h | 32 BYTEs | speedup factor for acceleration profile 1 (10h=1.0, 14h=1.25,20h=2.0, etc) |
| A4h | 32 BYTEs | speedup factor for acceleration profile 2 |
| C4h | 32 BYTEs | speedup factor for acceleration profile 3 |
| E4h | 32 BYTEs | speedup factor for acceleration profile 4 |
| 104h | 16 BYTEs | name of acceleration profile 1 (blank-padded) |
| 114h | 16 BYTEs | name of acceleration profile 2 (blank-padded) |
| 124h | 16 BYTEs | name of acceleration profile 3 (blank-padded) |
| 134h | 16 BYTEs | name of acceleration profile 4 (blank-padded) |

**Note:** unused bytes in the threshold speed fields are filled with 7Fh and unused bytes in the speedup factor fields are filled with 10h

## INT 33h Function 002Ch

MS MOUSE v7.0+

### *GET ACCELERATION PROFILES*

**Registers at call:**
AX = 002Ch

**Return Registers:**
AX = status (0000h success)
BX = currently-active acceleration profile
ES:SI -> acceleration profile data (see Table 21-12)

**See also:** Functions 002Bh, 002Dh, and 0033h

---

## INT 33h Function 002Dh

MS MOUSE v7.0+

### *SELECT ACCELERATION PROFILE*

**Registers at call:**
AX = 002Dh
BX = acceleration level
  0001h-0004h to set profile,
  or FFFFh to get current profile

**Return Registers:**
AX = status
  0000h successful
    ES:SI -> 16-byte blank-padded name of acceleration profile
  FFFEh invalid acceleration curve number
    ES:SI destroyed
BX = active acceleration curve number

**See also:** Functions 0013h, 002Bh, 002Ch, and 002Eh

---

## INT 33h Function 002Eh

MS MOUSE v8.10+

### *SET ACCELERATION PROFILE NAMES*

**Registers at call:**
AX = 002Eh
BL = flag (if nonzero, fill ES:SI buffer with default names on return)
ES:SI -> 64-byte buffer for profile names (16 bytes per name)

**Return Registers:**
AX = status (0000h success)
ES:SI buffer filled with default names if BL nonzero on entry

**Details:** This function is not supported by the Logitech driver v6.10.
**See also:** Functions 002Ch, 002Dh, 012Eh, and 022Eh

---

## INT 33h Function 002Fh

MS MOUSE v7.02+

### *MOUSE HARDWARE RESET*

**Registers at call:**
AX = 002Fh

**Return Registers:**
AX = status

**Details:** This function is invoked by the Microsoft mouse driver v8.20 on being called with INT 2Fh Function 530Bh.
**See also:** Function 0021h, INT 2Fh Function 53h (Chapter 19)

---

## INT 33h Function 0030h

MS MOUSE v7.04+

### *GET/SET BallPoint INFORMATION*

**Registers at call:**
AX = 0030h
CX = command
  0000h get status of BallPoint device
  other set rotation angle and masks
    BX = rotation angle (-32768 to 32767 degrees)
    CH = primary button mask
    CL = secondary button mask

**Return Registers:**
AX = button status (FFFFh if no BallPoint)
    bit 5: button 1
    bit 4: button 2
    bit 3: button 3
    bit 2: button 4
    other: zero
BX = rotation angle (0-360 degrees)
CH = primary button mask
CL = secondary button mask

### INT 33h Function 0031h

*MS MOUSE v7.05+*

### *GET CURRENT MINIMUM/MAXIMUM VIRTUAL COORDINATES*

**Registers at call:**
AX = 0031h

**Return Registers:**
AX = virtual X minimum
BX = virtual Y minimum
CX = virtual X maximum
DX = virtual Y maximum

**Details:** The minimum and maximum values are those set by AX=0007h and AX=0008h; the default is minimum = 0 and maximum = absolute maximum (see Function 0026h).

**See also:** Functions 0007h, 0008h, 0010h, and 0026h

### INT 33h Function 0032h

*MS MOUSE v7.05+*

### *GET ACTIVE ADVANCED FUNCTIONS*

**Registers at call:**
AX = 0032h

**Return Registers:**
AX = active function flags (FFFFh for v8.10)
    bit 15: function 0025h supported
    bit 14: function 0026h supported
    ...
    bit 0: function 0034h supported
BX = *unknown* (0000h) officially unused
CX = *unknown* (E000h) officially unused
DX = *unknown* (0000h) officially unused

**See also:** Functions 0025h to 0031h, 0033h, and 0034h

### INT 33h Function 0033h

*MS MOUSE v7.05+*

### *GET SWITCH SETTINGS AND ACCELERATION PROFILE DATA*

**Registers at call:**
AX = 0033h
CX = size of buffer
    0000h get required buffer size
    other
    ES:DX -> buffer of CX bytes

**Return Registers:**
AX = 0000h
    CX = required size (0154h for Logitech v6.10
    0159h for MS v8.10-8.20)

**Return Registers:**
AX = 0000h
    CX = number of bytes returned
    ES:DX buffer filled (see Table 21-13)

**See also:** Function 002Bh

**21-13 Format of switch settings/acceleration profile data buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | mouse type |
| 01h | BYTE | current language |
| 02h | BYTE | horizontal sensitivity (00h-64h) |
| 03h | BYTE | vertical sensitivity (00h-64h) |
| 04h | BYTE | double-speed threshold (00h-64h) |
| 05h | BYTE | ballistic curve (01h-04h) |
| 06h | BYTE | interrupt rate (01h-04h) |
| 07h | BYTE | cursor override mask |
| 08h | BYTE | laptop adjustment |
| 09h | BYTE | memory type (00h-02h) |
| 0Ah | BYTE | SuperVGA support (00h,01h) |
| 0Bh | BYTE | rotation angle |

| | | |
|---|---|---|
| 0Ch | BYTE | *unknown* |
| 0Dh | BYTE | primary button (01h-04h) |
| 0Eh | BYTE | secondary button (01h-04h) |
| 0Fh | BYTE | click lock enabled (00h,01h) |
| 10h | 324 BYTEs | acceleration profile data (see Table 21-12) |
| 154h | 5 BYTEs | *unknown* (Microsoft driver, but not Logitech) |

---

### INT 33h Function 0034h

*MS MOUSE v8.0+*

### *GET INITIALIZATION FILE*

**Registers at call:**
AX = 0034h

**Return Registers:**
AX = status (0000h successful)
ES:DX -> ASCIZ initialization (.INI) file name

---

### INT 33h Function 0035h

*MS MOUSE v8.10+*

### *LCD SCREEN LARGE POINTER SUPPORT*

**Registers at call:**
AX = 0035h
BX = function
  FFFFh get current settings
  other
    BH = style (00h normal, 01h reverse, 02h transparent)
    BL = size (00h small “1”, 01h medium “1.5”, 02h large “2”)
    CH = threshold (00h-64h)
    CL = active flag (00h disable size change, 01h enable)
    DX = delay (0000h-0064h)

**Return Registers:**
AX = 0000h
BH = style (see below)
BL = size (see below)
CH = threshold
CL = active flag (00h disabled, 01h enabled)
DX = delay
AX = 0000h

**Details:** This function is not supported by the Logitech driver v6.10.
**See also:** Functions 0012h and 002Ah

---

### INT 33h Function 004Dh

*MS MOUSE*

### *RETURN POINTER TO COPYRIGHT STRING*

**Registers at call:**
AX = 004Dh

**Return Registers:**
ES:DI -> copyright message “*** This is Copyright 1983 Microsoft”

**Details:** This function is also supported by Logitech, Kraft, Genius Mouse, and Mouse Systems mouse drivers.

In the Genius Mouse 9.06 driver, the ASCIZ signature “KYE” immediately follows the above copyright message (KYE Corp. manufactures the driver).

**See also:** Functions 0024h and 006Dh

---

### INT 33h Function 006Dh

*MS MOUSE*

### *GET VERSION STRING*

**Registers at call:**
AX = 006Dh

**Return Registers:**
ES:DI -> Microsoft version number of resident driver (see Table 21-14)

**Details:** This function is also supported by Logitech, Mouse Systems, Kraft, and Genius mouse drivers. The Mouse Systems 7.01 and Genius Mouse 9.06 drivers report their Microsoft version as 7.00 even though they do not support any of the functions from 0025h through 002Dh supported by the MS 7.00 driver (the Genius Mouse driver supports function 0026h, but it differs from the Microsoft function).
**See also:** Functions 0024h and 004Dh, Logitech Function 266Ch

## INT 33h Function 0034h

**21-14 Format of Microsoft version number:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | major version |
| 01h | BYTE | minor version (BCD) |

### INT 33h Function 012Eh
*MS MOUSE v8.10+*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 012Eh
BL = *unknown*

**Return Registers:**
AX = 0000h

**Details:** This function is not supported by the Logitech driver v6.10.

**See also:** Functions 002Eh and 022Eh

### INT 33h Function 022Eh
*MS MOUSE v8.10+*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 022Eh
BL = *unknown*

**Return Registers:**
AX = 0000h

**Details:** This function is not supported by the Logitech driver v6.10.

**See also:** Functions 002Eh and 012Eh

## Mouse Systems

### INT 33h Function 0042h
*PCMOUSE*

### *GET MSMOUSE STORAGE REQUIREMENTS*

**Registers at call:**
AX = 0042h

**Return Registers:**
AX = status
FFFFh successful
BX = buffer size in bytes for Functions 0050h and 0052h
0000h MSMOUSE not installed
0042h functions 0042h, 0050h, and 0052h not supported

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

**See also:** Functions 0050h and 0052h, Microsoft Function 0015h (above)

### INT 33h Function 0043h
*Mouse Systems MOUSE DRIVER v7.01*

### *CONFIGURE MOUSE*

**Registers at call:**
AX = 0043h
CX:BX -> *unknown* buffer (see Table 21-15)
DL = *unknown*

**Return Registers:**
*unknown*

**Details:** This function also calls routines for INT 33h Function 0053h and INT 33h Function 004Fh.

This function is also supported by the Genius Mouse 9.06 driver.

**21-15 Format of buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | I/O port address |
| 02h | BYTE | *unknown* |
| 03h | BYTE | interrupt number |
| 04h | BYTE | interrupt mask for interrupt controller |
| 05h | 5 BYTEs | *unknown* |

---

**INT 33h Function 0044h** *Mouse Systems MOUSE DRIVER v7.01*

***TOGGLE UNKNOWN FLAG***

**Registers at call:**
AX = 0044h
CX = CDEFh

**Return Registers:**
AX = new state of *unknown*

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

**See also:** Function 0045h

---

**INT 33h Function 0045h** *Mouse Systems MOUSE DRIVER v7.01*

***TOGGLE UNKNOWN FLAG***

**Registers at call:**
AX = 0045h
CX = CDEFh

**Return Registers:**
AX = new state of *unknown*

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

**See also:** Function 0044h

---

**INT 33h Function 0047h** *Mouse Systems MOUSE DRIVER v7.01*

***SET BUTTON ASSIGNMENTS***

**Registers at call:**
AX = 0047h
ES:BX -> button assignments
(3 bytes, combinations of "L", "M", "R")

**Return Registers:**
*unknown*

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

**See also:** Function 0067h

---

**INT 33h Function 0048h** *Mouse Systems MOUSE DRIVER v7.01*

***GET UNKNOWN VALUE***

**Registers at call:**
AX = 0048h
BX = CDEFh

**Return Registers:**
CX = *unknown*
BH = *unknown*
BL = *unknown* (if 50h, driver is using PS/2 pointing device BIOS interface)

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

---

**INT 33h Function 004Ch** *Mouse Systems MOUSE DRIVER v7.01*

***SET UNKNOWN FLAG***

**Registers at call:**
AX = 004Ch
BX = CDEFh

**Return Registers:**
none

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

**See also:** Function 006Ch

---

**INT 33h Function 004Fh** *Mouse Systems MOUSE DRIVER v7.01*

***ENABLE MOUSE***

**Registers at call:**
AX = 004Fh

**Return Registers:**
none

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

**See also:** Functions 0043h and 0053h

---

**INT 33h Function 0050h** *PCMOUSE*

***SAVE MSMOUSE STATE***

**Registers at call:**
AX = 0050h
BX = buffer size (ignored by some driver versions)
ES:DX -> buffer

**Return Registers:**
AX = FFFFh if successful

**Details:** The buffer must be large enough to hold the entire state, or following data will be overwritten by state data in versions which ignore BX; use INT 33h Function 0042h to get the required size.

This function is also supported by the Genius Mouse 9.06 driver.

**See also:** Functions 0042h and 0052h

---

**INT 33h Function 0052h** *PCMOUSE*

***RESTORE MSMOUSE STATE***

**Registers at call:**
AX = 0052h
BX = buffer size (ignored by some driver versions)
ES:DX -> buffer

**Return Registers:**
AX = FFFFh if successful

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

**See also:** Function 0050h

---

**INT 33h Function 0053h** *Mouse Systems MOUSE DRIVER v7.01*

***DISABLE MOUSE***

**Registers at call:**
AX = 0053h

**Return Registers:**
none

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

**See also:** Functions 0043h and 004Fh

---

**INT 33h Function 0054h** *Mouse Systems MOUSE DRIVER v7.01*

***SELECT ULTRARES ACCELERATION LEVEL***

**Registers at call:**
AX = 0054h
CX = CDEFh
BX = *acceleration level?* (NOP if <= 0000h)

**Return Registers:**
*unknown*

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

**See also:** Function 005Ah

---

**INT 33h Function 0058h** *Mouse Systems MOUSE DRIVER v7.01*

***UNDETERMINED FUNCTION***

**Registers at call:**
AX = 0058h

**Return Registers:**
AX = CS of driver
CX:BX = original INT 33h vector
DX = *unknown*

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

---

**INT 33h Function 005Ah** *Mouse Systems MOUSE DRIVER v7.01*

***SET ULTRARES ACCELERATIONS***

**Registers at call:**
AX = 005Ah
CX = number of WORDs to copy (max 0014h, but not range-checked)

DX:SI -> buffer containing *thresholds* (CX words)
DX:BX -> buffer containing *acceleration values* (9*14h words, only first CX of each 14h used)
*others, if any, unknown*

**Return Registers:**
CF clear
*others, if any, unknown*

**Details:** This function is also supported by the Genius Mouse 9.06 driver.
**See also:** Function 0054h

---

### INT 33h Function 0061h
### *UNDETERMINED FUNCTION*

*Mouse Systems MOUSE DRIVER v7.01*

**Registers at call:**
AX = 0061h
BX = CDEFh

**Return Registers:**
CX = *unknown*

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

### INT 33h Function 0067h
### *GET MOUSE BUTTONS*

*Mouse Systems MOUSE DRIVER v7.01*

**Registers at call:**
AX = 0067h

**Return Registers:**
BL = *number of buttons*

**Details:** This function is also supported by the Genius Mouse 9.06 driver.
**See also:** Function 0047h

### INT 33h Function 006Ch
### *CLEAR UNKNOWN FLAG*

*Mouse Systems MOUSE DRIVER v7.01*

**Registers at call:**
AX = 006Ch
BX = CDEFh

**Return Registers:**
none

**Details:** This function is also supported by the Genius Mouse 9.06 driver.
**See also:** Function 004Ch

### INT 33h Function 0070h
### *INSTALLATION CHECK*

*Mouse Systems POPUP.COM*

**Registers at call:**
AX = 0070h
BX = ABCDh

**Return Registers:**
AX = ABCDh if installed
BX:CX -> data structure (see Table 21-16)

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

The v7.01 POPUP.COM and menu drivers also check for the signature CDh ABh BAh DCh at offset -2Ch from the interrupt handler.

If POPUP is not loaded, the returned data structure contains the proper signature at offset 00h, but not at offset 08h.

**21-16 Format of data structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | signature ABCDh |
| 02h | DWORD | pointer to *info structure* (see Table 21-17) |
| 06h | 2 BYTEs | *unknown* |
| 08h | WORD | signature ABCDh |

**21-17 Format of info structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | driver version |
| 02h | 8 BYTEs | *unknown* |
| 0Ah | WORD | segment of *unknown* |
|  |  | *additional fields, if any, unknown* |

### INT 33h Function 0072h
### *UNDETERMINED FUNCTION*

*Mouse Systems*
*MOUSE DRIVER v7.01*

**Registers at call:**
AX = 0072h
BX = ABCDh

**Return Registers:**
*unknown*

**Details:** This function is also supported by the Genius Mouse 9.06 driver.

---

### INT 33h Function 0073h
### *GET BUTTON ASSIGNMENTS*

*Mouse Systems*
*MOUSE DRIVER v7.01*

**Registers at call:**
AX = 0073h
BX = CDEFh
ES:DX -> 3-byte buffer for button assignments

**Return Registers:**
CX = *number of buttons*
ES:DX buffer filled (default is "LMR")

**Details:** This function is also supported by the Genius Mouse 9.06 driver.
**See also:** Function 0067h

## Logitech

### INT 33h Function 136Ch
### *UNDETERMINED FUNCTION*

*LOGITECH MOUSE v6.10+*

**Registers at call:**
AX = 136Ch
BX = *unknown*

**Return Registers:**
AX = *unknown*
BX = *unknown*

---

### INT 33h Function 146Ch
### *GET/SET UNKNOWN VALUE*

*LOGITECH MOUSE v6.10+*

**Registers at call:**
AX = 146Ch
BL = function
    00h set *unknown*
        BH = new value (zero/nonzero to clear/set)
    else get *unknown*

**Return Registers:**
*unknown*

---

### INT 33h Function 156Ch
### *GET SIGNATURE AND VERSION STRINGS*

*LOGITECH MOUSE v6.10+*

**Registers at call:**
AX = 156Ch

**Return Registers:**
ES:DI -> signature "LOGITECH MOUSE DRIVER"
ES:SI -> version string, terminated with CRLF

**See also:** Microsoft Function 006Dh (above)

---

### INT 33h Function 166Ch
### *UNDETERMINED FUNCTION*

*LOGITECH MOUSE v6.10+*

**Registers at call:**
AX = 166Ch
BL = *unknown*
    00h *unknown*
    01h *unknown*
    other *unknown*
        BH = new value of *unknown*

**Return Registers:**
AX = FFFFh

---

### INT 33h Functions 176Ch, 186Ch, and 196Ch
*LOGITECH MOUSE v6.10+*

### *UNDETERMINED FUNCTIONS*

**Registers at call:**
AX = 176Ch, 186Ch, and 196Ch
*others, if any, unknown*

**Return Registers:**
*unknown*

---

### INT 33h Function 1A6Ch
*LOGITECH MOUSE v6.10+*

### *GET UNKNOWN VALUES*

**Registers at call:**
AX = 1A6Ch

**Return Registers:**
AX = FFFFh
BX = *unknown*
CX = *unknown*

**See also:** Function 1B6Ch

---

### INT 33h Function 1B6Ch
*LOGITECH MOUSE v6.10+*

### *SET UNKNOWN VALUE*

**Registers at call:**
AX = 1B6Ch
BX = new value for *unknown* (0000h-0003h)

**Return Registers:**
AX = FFFFh

**See also:** Function 1A6Ch

---

### INT 33h Function 1C6Ch
*LOGITECH MOUSE v6.10+*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 1C6Ch
BX = *unknown*
  <42h *unknown*
  =42h *unknown*
  >42h *unknown*
    ES:DI -> *unknown*

**Return Registers:**
AX = *unknown*

---

### INT 33h Function 1D6Ch
*LOGITECH MOUSE*

### *GET COMPASS PARAMETER*

**Registers at call:**
AX = 1D6Ch

**Return Registers:**
BX = direction (0=north, 1=south, 2=east, 3=west)

**See also:** Function 1E6Ch

---

### INT 33h Function 1E6Ch
*LOGITECH MOUSE*

### *SET COMPASS PARAMETER*

**Registers at call:**
AX = 1E6Ch
BX = direction (0=north, 1=south, 2=east, 3=west)

**Return Registers:**
none

**See also:** Function 1D6Ch

---

### INT 33h Function 1F6Ch
*LOGITECH MOUSE*

### *GET BALLISTICS INFORMATION*

**Registers at call:**
AX = 1F6Ch

**Return Registers:**
BX = 0=off, 1=on
CX = 1=low, 2=high

**See also:** Function 236Ch, Microsoft Function 002Ch (above)

## INT 33h Functions 176Ch, 186Ch, and 196Ch

**INT 33h Function 206Ch** *LOGITECH MOUSE*

***SET LEFT OR RIGHT PARAMETER***

**Registers at call:**
AX = 206Ch
BX = parameter (00h = right, FFh = left)

**Return Registers:**
none

**See also:** Function 216Ch

---

**INT 33h Function 216Ch** *LOGITECH MOUSE*

***GET LEFT OR RIGHT PARAMETER***

**Registers at call:**
AX = 216Ch

**Return Registers:**
BX = parameter (00h = right, FFh = left)

**See also:** Function 206Ch

---

**INT 33h Function 226Ch** *LOGITECH MOUSE*

***REMOVE DRIVER FROM MEMORY***

**Registers at call:**
AX = 226Ch

**Return Registers:**
none

**Details:** This function only frees memory; it does not restore hooked interrupts.

---

**INT 33h Function 236Ch** *LOGITECH MOUSE*

***SET BALLISTICS INFORMATION***

**Registers at call:**
AX = 236Ch
BX = 0=off, 1=on
CX = 1=low, 2=high

**Return Registers:**
none

**See also:** Function 1F6Ch, Microsoft Function 002Ch

---

**INT 33h Function 246Ch** *LOGITECH MOUSE*

***GET PARAMETERS AND RESET SERIAL MOUSE***

**Registers at call:**
AX = 246Ch
ES:DX -> parameter table buffer
(see Table 21-18)

**Return Registers:**
AX = FFFFh if driver installed for serial mouse

**See also:** Function 256Ch Subfunctions 0000h, 0001h, and 0003h; Microsoft Function 0000h

**21-18 Format of parameter table:**

| *Offset* | *Size* | *Description* | |
|---|---|---|---|
| 00h | WORD | baud rate divided by 100 | (serial mouse only) |
| 02h | WORD | emulation | (serial mouse only) |
| 04h | WORD | report rate | (serial mouse only) |
| 06h | WORD | firmware revision | (serial mouse only) |
| 08h | WORD | 00h | (serial mouse only) |
| 0Ah | WORD | port | (serial mouse only) |
| 0Ch | WORD | physical buttons | |
| 0Eh | WORD | logical buttons | |

---

**INT 33h Function 256Ch, Subfunction 0000h** *LOGITECH MOUSE*

***SET PARAMETERS - SET BAUD RATE (SERIAL MOUSE ONLY)***

**Registers at call:**
AX = 256Ch
BX = 0000h
CX = rate (0=1200, 1=2400, 2=4800, 3=9600)

**Return Registers:**
AX = FFFFh if driver installed for serial mouse

**See also:** Function 246Ch, Function 256Ch Subfunctions 0001h and 0002h, Function 276Ch

### INT 33h Function 256Ch, Subfunction 0001h

*LOGITECH MOUSE*

### *SET PARAMETERS - SET EMULATION (SERIAL MOUSE ONLY)*

Registers at call:
AX = 256Ch
BX = 0001h
CX = emulation (see Table 21-19)

Return Registers:
AX = FFFFh if driver installed for serial mouse

**See also:** Function 246Ch, Function 256Ch Subfunctions 0000h and 0003h, Function 276Ch

**21-19 Logitech mouse emulation types:**

| | |
|---|---|
| 00h | 5 byte packed binary |
| 01h | 3 byte packed binary |
| 02h | hexadecimal |
| 03h | relative bit pad |
| 04h | not supported |
| 05h | MM Series |
| 06h | not supported |
| 07h | Microsoft |

### INT 33h Function 256Ch, Subfunction 0002h

*LOGITECH MOUSE*

### *SET PARAMETERS - SET REPORT RATE (SERIAL MOUSE ONLY)*

Registers at call:
AX = 256Ch
BX = 0002h
CX = rate (0=10, 1=20, 2=35, 3=50, 4=70, 5=100, 6=150)

Return Registers:
AX = FFFFh if driver installed for serial mouse

**See also:** Function 246Ch, Function 256Ch Subfunctions 0001h and 0003h, Function 276Ch

### INT 33h Function 256Ch, Subfunction 0003h

*LOGITECH MOUSE*

### *SET PARAMETERS - SET MOUSE PORT (SERIAL MOUSE ONLY)*

Registers at call:
AX = 256Ch
BX = 0003h
CX = port (1, 2)

Return Registers:
AX = FFFFh if driver installed for serial mouse

**See also:** Function 246Ch, Function 256Ch Subfunctions 0000h and 0004h, Function 276Ch

### INT 33h Function 256Ch, Subfunction 0004h

*LOGITECH MOUSE*

### *SET PARAMETERS - SET MOUSE LOGICAL BUTTONS*

Registers at call:
AX = 256Ch
BX = 0004h
CX = buttons (2, 3)

Return Registers:
AX = FFFFh if driver installed for serial mouse

**See also:** Functions 246Ch and 276Ch

### INT 33h Function 266Ch

*LOGITECH MOUSE*

### *GET VERSION*

Registers at call:
AX = 266Ch

Return Registers:
BX = 'SS'
CH = '4' major version number
CL = '1' minor version number

**See also:** Function 006Dh

### INT 33h Function 276Ch

*LOGITECH MOUSE*

### *UNKNOWN (Tries MMSeries, Baud 2400)*

Registers at call:
AX = 276Ch

Return Registers:
none

**See also:** Function 256Ch

---

### INT 33h Function 4F00h
*LOGITECH MOUSE v6.10+*

***GET UNKNOWN DATA***

**Registers at call:**
AX = 4F00h

**Return Registers:**
AX = 004Fh if supported
BX = *unknown*
ES:DI -> *unknown*

**See also:** Function 4F01h

---

### INT 33h Function 4F01h
*LOGITECH MOUSE v6.10+*

***UNDETERMINED FUNCTION***

**Registers at call:**
AX = 4F01h
ES = *unknown*

**Return Registers:**
AX = 004Fh if supported
ES:DI -> *unknown*

**See also:** Function 4F00h

## Smooth Mouse Driver

SMD is a programmer's library by Andy Hakim which provides a graphics-style mouse cursor in text mode. PrecisePoint is an SMD-based TSR which replaces the block mouse cursor in text applications, and supports all of the SMD calls.

---

### INT 33h Function 3000h
*Smooth Mouse Driver*

***INSTALLATION CHECK***

**Registers at call:**
AX = 3000h

**Return Registers:**
AX = FFFFh if installed
BX = version number (BH = major, BL = minor)

**See also:** Functions 3001h and 3003h, Microsoft Function 0000h (above)

---

### INT 33h Function 3001h
*Smooth Mouse Driver*

***ENABLE SMOOTH MOUSE***

**Registers at call:**
AX = 3001h

**Return Registers:**
AX = status (0000h = disabled, 0001h = enabled)

**Details:** SMD remains disabled if running under Desqview or in graphics mode.

**See also:** Function 3002h, Microsoft Functions 0001h and 0002h (above)

---

### INT 33h Function 3002h
*Smooth Mouse Driver*

***DISABLE SMOOTH MOUSE***

**Registers at call:**
AX = 3002h

**Return Registers:**
AX = status (0000h = disabled, 0001h = enabled)

**See also:** Functions 3000h and 3001h, Microsoft Functions 0001h and 0002h (above)

---

### INT 33h Function 3003h
*Smooth Mouse Driver*

***GET INFORMATION***

**Registers at call:**
AX = 3003h
BL = data structure selector
    00h Primary Bitmap (used for 25 line mode)
    01h Secondary Bitmap (used for 43/50 line modes)
    02h Sacrifice Character Map
    03h Program Information

**Return Registers:**
ES:DX -> selected data structure

**See also:** Function 3000h

#### 21-20 Format of Primary/Secondary Bitmap [SMD_BITMAP_STRUCT]

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | vertical size of bitmap (00h - 10h) |
| 01h | BYTE | horizontal size of bitmap (00h - 10h) |

| | | |
|---|---|---|
| 02h | BYTE | vertical hotspot position (00h - 10h) |
| 03h | BYTE | horizontal hotspot position (00h - 10h) |
| 04h | 16 WORDs | cursor mask bitmap data |
| 14h | 16 WORDs | screen mask bitmap data |

### 21-21 Format of Sacrifice Character Map [SMD_SMAP_STRUCT]

The Sacrifice Character Map consists of nine bytes, each a character value to be used in place of the actual character for the corresponding position on the screen occupied by part or all of the mouse cursor, as shown in the following diagram:

```
+-----------+
| 0 | 1 | 2 |
|---+---+---|
| 3 | 4 | 5 |
|---+---+---|
| 6 | 7 | 8 |
+-----------+
```

SMD redefines the active font such that these nine characters appear as the actual character with the appropriate portion of the mouse cursor overlaid on top.

### 21-22 Format of Program Information [SMD_INFO_STRUCT]

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of old interrupt 33h handler |
| 02h | WORD | offset of old interrupt 33h handler |
| 04h | WORD | PSP of SMD |
| 06h | BYTE | ENABLE/DISABLE manual setting status |
| 07h | BYTE | ENABLE/DISABLE internal usage status |

---

**INT 33h Function 3004h** *Smooth Mouse Driver*

***RESERVED FOR FUTURE EXPANSION***

| Registers at call: | Return Registers: |
|---|---|
| AX = 3004h | none |

**See also:** Function 3000h

---

**INT 33h Function 3005h** *Smooth Mouse Driver*

***RESERVED FOR FUTURE EXPANSION***

| Registers at call: | Return Registers: |
|---|---|
| AX = 3005h | none |

**See also:** Function 3000h

## Quick Mouse Reset

QMR (Cove Software's Quick Mouse Reset) monitors the mouse service interrupt (int 33h) and substitutes a fast software reset (mouse Function 0021h) for the slow hardware reset (mouse Function 0000h). QMR may use any multiplex number between C0h and FFh; it searches for itself in that range, and installs using the first free multiplex number in the range if not already loaded. As with any TSR which scans this range of multiplex numbers, there are many potential conflicts, which are listed in Table 2-2.

---

**INT 2Fh Function C000h** *QMR*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = C000h | AL = FFh if installed<br>ES:DI -> signature string "QMR1!" |

**Conflicts:** see Table 2-2

**See also:** Functions C001h to C003h

---

### INT 2Fh Function C001h *QMR*

***REQUEST HARDWARE RESET***

**Purpose:** This function is used to force a full mouse reset when QMR is installed.

**Registers at call:**
AX = C001h

**Return Registers:**
ES = QMR code segment
AL destroyed

**Conflicts:** see Table 2-2

**See also:** Functions C000h and C002h, Microsoft Mouse INT 33h Function 0000h (above)

---

### INT 2Fh Function C002h *QMR*

***DISABLE QMR***

**Purpose:** This call temporarily disables QMR.

**Registers at call:**
AX = C002h

**Return Registers:**
ES = QMR code segment
AL destroyed

**Conflicts:** see Table 2-2

**See also:** Functions C000h and C003h

---

### INT 2Fh Function C003h *QMR*

***ENABLE QMR***

**Purpose:** This call enables QMR after it has been disabled.

**Registers at call:**
AX = C003h

**Return Registers:**
ES = QMR code segment
AL destroyed

**Conflicts:** see Table 2-2

**See also:** Functions C000h and C002h

## Miscellaneous

### INT 15h Function 00h *Amstrad PC1512*

***GET AND RESET MOUSE COUNTS***

**Registers at call:**
AH = 00h

**Return Registers:**
CX = signed X count
DX = signed Y count

**Conflicts:** Cassette (Chapter 4), MultiDOS Plus (Chapter 45), VMiX (Chapter 46)

**See also:** Microsoft INT 33h Function 000Bh (above)

---

### INT 2Fh Function 530Bh *unknown (MOUSEPWR.COM, others) (undocumented)*

***UNDETERMINED FUNCTION***

**Registers at call:**
AX = 530Bh
BX = subfunction
    0003h *unknown*
    0004h *unknown*
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** It appears that subfunction 0003h reads or restores the current mouse settings (the MS Mouse driver hooks AX=530Bh), and 0004h might be the converse.

**See also:** POWER.EXE Function 5400h (Chapter 19)

---

### INT 2Fh Function D000h *ZWmous*

***INSTALLATION CHECK***

**Registers at call:**
AX = D000h

**Return Registers:**
AX = 5A57h ("ZW") if installed
    BX = segment of resident code

**Program:** ZWmous is a shareware TSR by Zen Wu which permits the use of a mouse with many non-mouse applications by entering the letter under the mouse cursor on button presses.
**Conflicts:** MDEBUG (Chapter 57), Lotus CD/Networker (NI); see also Table 2-2
**See also:** Microsoft Mouse INT 33h Function 0003h (above)

---

### INT 33h Function 0010h — *Genius MOUSE*

### *DEFINE SCREEN REGION FOR UPDATING*

**Registers at call:**
AX = 0010h
ES:DX -> update region list (see Table 21-23)

**Return Registers:**
none

**Details:** The mouse cursor is hidden in the specified region, and needs to be explicitly turned on again. This version of the call is described in an August 1988 version of the Genius Mouse programmer's reference; it has been changed to conform to the Microsoft version shown above by version 9.06 (and possibly earlier versions).
**Conflicts:** Microsoft Mouse Function 0010h (above)
**See also:** Microsoft Mouse Functions 0001h, 0002h, 0007h, and 0010h (above)

**21-23 Format of update region list:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | left-most column |
| 02h | WORD | top-most row |
| 04h | WORD | right-most column |
| 06h | WORD | bottom-most row |

---

### INT 33h Function 0011h — *Genius Mouse 9.06*

### *GET NUMBER OF BUTTONS*

**Registers at call:**
AX = 0011h

**Return Registers:**
AX = FFFFh
BX = number of buttons

**See also:** Microsoft Mouse Function 0000h (above), Mouse Systems Fucntion 0067h (above), Logitech Function 246Ch (above)

---

### INT 33h Function 0026h — *Genius Mouse 9.06*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 0026h

**Return Registers:**
CX = 0204h if CX was 0105h on entry, else unchanged

**Conflicts:** Microsoft Mouse Function 0026h (above)

---

### INT 33h Function 004Bh — *Z-NIX MOUSE DRIVER v7.04d*

### *INSTALLATION CHECK*

**Registers at call:**
AX = 004Bh

**Return Registers:**
ES:DI -> signature/description string if installed

**Details:** The signature string in v7.04d is "Z-NIX;BUS,AUX,Serial 3-byte and 5-byte Mouse Driver;ZMOUSE;v7.04d".

---

### INT 33h Function 0055h — *Kraft Mouse*

### *GET UNKNOWN VALUES*

**Registers at call:**
AX = 0055h

**Return Registers:**
CX = *unknown*
DX = *unknown*
ES = *unknown*

## CHAPTER 22

# Sound and Sound Boards

With the rise of MultiMedia presentations, sound generation devices have become a significant feature in most modern systems. Unfortunately the field is so new that generally accepted standards have not yet become the norm. This chapter describes the individual APIs of all sound boards that we have been able to locate.

## Ad Lib

---

**INT 65h** *Ad Lib SOUND.COM*

***INTERFACE***

**Registers at call:**

SI = function number (see also entries below)

- 0000h Init
- 0002h RelTimeStart
- 0003h SetState
- 0004h GetState
- 0005h Flush
- 0006h SetMode
- 0007h GetMode
- 0008h SetRelVolume
- 0009h SetTempo
- 000Ah SetTranspose
- 000Bh GetTranspose
- 000Ch SetActVoice
- 000Dh GetActVoice
- 000Eh PlayNoteDel
- 000Fh PlayNote
- 0010h SetTimbre
- 0011h SetPitch
- 0012h SetTickBeat
- 0013h NoteOn
- 0014h NoteOff
- 0015h Timbre
- 0016h SetPitchBend
- 0017h WaveForm

ES:BX -> arguments

**Return Registers:**

vary by function

**Details:** The installation check consists of checking for the signature block immediately preceding the interrupt handler (see Table 22-1).
**See also:** Function 8000h

**22-1 Format of signature block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | version number |
| 02h | 19 BYTEs | "SOUND-DRIVER-AD-LIB" |
| 15h | BYTE | 01h |
| 16h | BYTE | 01h |
| 17h | BYTE | 00h |

---

### INT 65h Function 0000h
*Ad Lib SOUND.COM*

### *INITIALIZE (RESET)*

**Registers at call:**
SI = 0000h

**Return Registers:**
none

**Conflicts:** see Table 2-3

---

### INT 65h Function 0003h
*Ad Lib SOUND.COM*

### *SET STATE*

**Registers at call:**
SI = 0003h
ES:BX -> WORD state
(0000h disabled, 0001h enabled)

**Return Registers:**
none

**Conflicts:** see Table 2-3
**See also:** Function 0004h

---

### INT 65h Function 0004h
*Ad Lib SOUND.COM*

### *GET STATE*

**Registers at call:**
SI = 0004h

**Return Registers:**
AX = state
0000h all done playing sounds
else still playing sounds

**Conflicts:** see Table 2-3
**See also:** Function 0003h

---

### INT 65h Function 0006h
*Ad Lib SOUND.COM*

### *SET MODE*

**Registers at call:**
SI = 0006h
ES:BX -> WORD mode
(0000h melodic, 0001h percussive)

**Return Registers:**
none

**Conflicts:** see Table 2-3
**See also:** Function 0007h

---

### INT 65h Function 0007h
*Ad Lib SOUND.COM*

### *GET MODE*

**Registers at call:**
SI = 0007h

**Return Registers:**
AX = mode
0000h melodic
0001h percussive

**Conflicts:** see Table 2-3
**See also:** Function 0006h

---

### INT 65h Function 000Ch
*Ad Lib SOUND.COM*

***SET ACTIVE VOICE***

**Registers at call:**
SI = 000Ch
ES:BX -> WORD voice = 0000h to 0008h

**Return Registers:**
none

**Conflicts:** see Table 2-3
**See also:** Function 000Dh

---

### INT 65h Function 000Dh
*Ad Lib SOUND.COM*

***GET ACTIVE VOICE***

**Registers at call:**
SI = 000Dh

**Return Registers:**
AX = voice (0000h to 0008h)

**Conflicts:** see Table 2-3
**See also:** Function 000Ch

## SoundBlaster

This section describes the API for the SoundBlaster Card produced by Creative Labs, Inc.

---

### INT 2Fh Function FBFBh
*SoundBlaster speech driver*
*(undocumented)*

***INSTALLATION CHECK***

**Registers at call:**
AX = FBFBh
ES = 0000h

**Return Registers:**
ES nonzero if installed
ES:BX -> entry point data structure (see Table 22-2

**Conflicts:** Borland DOS Extenders (Chapter 36); see also Table 2-2
**See also:** INT 80h Function 0000h, INT F3h

**22-2 Format of entry point data structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 3 BYTEs | signature "FB " |
| 03h | BYTE | *driver major version number* |
| 04h | DWORD | speech driver entry point (see Table 22-3) |
| 08h | 24 BYTEs | *unknown* |
| 20h | ? BYTEs | data buffer for calling speech driver |

**22-3 driver entry point calling convention:**

**Registers at call:**
AL = function
07h speak a string
data buffer (see Table 22-2) contains:
BYTE length of string
N BYTEs string to speak

**Return Registers:**
none

---

### INT 80h Function 0000h
*SoundBlaster SBFM driver*

***GET VERSION***

**Registers at call:**
BX = 0000h

**Return Registers:**
*unknown*

**Details:** SBFM installs at a free interrupt in the range 80h through BFh.
**Conflicts:** see Index for interrupts 80h through BFh for potential conflicts
**See also:** Function 0008h, INT 2Fh Function FBFBh

**INT 80h Function 0001h** *SoundBlaster SBFM driver*

***SET MUSIC STATUS BYTE ADDRESS***

| **Registers at call:** | **Return Registers:** |
|---|---|
| BX = 0001h<br>DX:AX -> music status byte | none |

**See also:** Functions 0000h, 0002h, and 0003h

**INT 80h Function 0002h** *SoundBlaster SBFM driver*

***SET INSTRUMENT TABLE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| BX = 0002h<br>CX = number of instruments<br>DX:AX -> instrument table | none |

**See also:** Functions 0000h, 0001h, and 0005h

**INT 80h Function 0003h** *SoundBlaster SBFM driver*

***SET SYSTEM CLOCK RATE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| BX = 0003h<br>AX = clock rate divisor (1193180 / desired frequency in Hertz)<br>FFFFh to restore to 18.2 Hz | none |

**See also:** Functions 0000h, 0001h, and 0004h

**INT 80h Function 0004h** *SoundBlaster SBFM driver*

***SET DRIVER CLOCK RATE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| BX = 0004h<br>AX = driver clock rate divisor (1193180 / frequency in Hertz) | none |

**Details:** The default frequency is 96 Hz.

**See also:** Functions 0000h and 0003h

**INT 80h Function 0005h** *SoundBlaster SBFM driver*

***TRANSPOSE MUSIC***

| **Registers at call:** | **Return Registers:** |
|---|---|
| BX = 0005h<br>AX = semi-tone offset | none |

**See also:** Functions 0000h, 0002h, and 0006h

**INT 80h Function 0006h** *SoundBlaster SBFM driver*

***PLAY MUSIC***

| **Registers at call:** | **Return Registers:** |
|---|---|
| BX = 0006h<br>DX:AX -> music block | AX = status<br>0000h successful<br>0001h music already active |

**See also:** Functions 0000h, 0007h, and 000Ah; SND INT 1Ah Function FF04h (below)

**INT 80h Function 0007h** *SoundBlaster SBFM driver*

***STOP MUSIC***

| **Registers at call:** | **Return Registers:** |
|---|---|
| BX = 0007h | AX = status<br>0000h successful<br>0001h music not active |

**See also:** Functions 0000h, 0006h, and 0009h; SND INT 1Ah Function FF05h (below)

**INT 80h Function 0001h**

---

**INT 80h Function 0008h** *SoundBlaster SBFM driver*

***RESET DRIVER***

**Registers at call:**
BX = 0008h

**Return Registers:**
AX = status
0000h successful
0001h music is active

**See also:** Function 0000h

---

**INT 80h Function 0009h** *SoundBlaster SBFM driver*

***PAUSE MUSIC***

**Registers at call:**
BX = 0009h

**Return Registers:**
AX = status
0000h successful
0001h no music active

**See also:** Functions 0000h, 0007h, and 000Ah; SND INT 1Ah Function FF01h (below)

---

**INT 80h Function 000Ah** *SoundBlaster SBFM driver*

***RESUME MUSIC***

**Registers at call:**
BX = 000Ah

**Return Registers:**
AX = status
0000h successful
0001h no music paused

**See also:** Functions 0000h, 0006h, and 0009h

---

**INT 80h Function 000Bh** *SoundBlaster SBFM driver*

***SET USER-DEFINED TRAP FOR SYSTEM-EXCLUSIVE COMMANDS***

**Registers at call:**
BX = 000Bh
DX:AX -> trap routine

**Return Registers:**
none

**See also:** Functions 0000h

---

**INT F3h** *SoundBlaster*

***POINTER TO ECHO VALUE***

**Details:** This is not a vector, but a pointer to a DWORD containing the echo value selected with SET-ECHO.EXE.

**See also:** INT 2Fh Function FBFBh

## PCM driver

PCM.COM is a sound driver for Media Vision's Pro Audio Spectrum sound boards which provides a superset of the standard PCM driver's capabilities (see the "Media Vision" section for PCM.COM-specific functions).

---

**INT 94h;Function 0000h** *PCM driver (partially doc.)*

***INITIALIZE SOUND***

**Registers at call:**
SI = 0000h
ES:BX -> parameters

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)

**Details:** The installation check consists of testing for the signature string "PCMDRIVER" immediately preceding the interrupt handler; the word preceding the signature gives the PCM driver's version.

**See also:** Functions 0001h, 0002h, 0003h, 0004h, 0005h, and 000Ah

---

**INT 94h Function 0001h** PCM driver (partially doc.)

***INITIALIZE PCM***

**Registers at call:**
SI = 0001h
ES:BX -> parameters

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Functions 0000h, 0002h, 0003h, and 000Ah

---

**INT 94h Function 0002h** PCM driver (partially doc.)

***INITIALIZE PCM INFO***

**Registers at call:**
SI = 0002h
ES:BX -> initialization parameters (see Table 22-4)

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Functions 0000h, 0001h, 0003h, 000Ah

**22-4 Format of initialization parameters:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | rate |
| 04h | WORD | channel number |
| 06h | WORD | "comp" |
| 08h | WORD | "dsize" |

---

**INT 94h Function 0003h** PCM driver (partially doc.)

***INITIALIZE DMA BUFFER***

**Registers at call:**
SI = 0003h
ES:BX -> parameters (see Table 22-5)

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Functions 0000h, 000Ah, and 000Bh

**22-5 Format of DMA initialization parameters:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to DMA buffer |
| 04h | WORD | size of DMA buffer |
| 06h | WORD | number of divisions |

---

**INT 94h Function 0004h** PCM driver (partially doc.)

***INITIALIZE USER FUNCTION***

**Registers at call:**
SI = 0004h
ES:BX -> parameters (see Table 22-6)

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Functions 0000h and 0001h

**22-6 Format of user function parameters:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to user function |

---

**INT 94h Function 0005h** PCM driver (partially doc.)

***BEGIN AUDIO PLAY***

**Registers at call:**
SI = 0005h

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Functions 0000h, 0006h, 0007h, and 0009h

**INT 94h Function 0006h** *PCM driver*
***BEGIN AUDIO RECORD*** *(partially doc.)*

**Registers at call:** **Return Registers:**
SI = 0006h *unknown*
**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Functions 0005h, 0007h, and 0009h

**INT 94h Function 0007h** *PCM driver*
***PAUSE AUDIO PLAY/RECORD*** *(partially doc.)*

**Registers at call:** **Return Registers:**
SI = 0007h *unknown*
**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Functions 0005h, 0006h, 0008h

**INT 94h Function 0008h** *PCM driver*
***RESUME AUDIO PLAY/RECORD*** *(partially doc.)*

**Registers at call:** **Return Registers:**
SI = 0008h *unknown*
**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Functions 0007h

**INT 94h Function 0009h** *PCM driver*
***STOP AUDIO PLAY/RECORD*** *(partially doc.)*

**Registers at call:** **Return Registers:**
SI = 0009h *unknown*
**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Functions 0005h, 0006h, and 0007h

**INT 94h Function 000Ah** *PCM driver*
***UNHOOK INTERRUPTS AND TURN OFF DMA*** *(partially doc.)*

**Registers at call:** **Return Registers:**
SI = 000Ah *unknown*
**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Functions 0000h, 0001h, and 0003h

**INT 94h Function 000Bh** *PCM driver*
***FIND VALID DMA BUFFER IN HUGE MEMORY BLOCK*** *(partially doc.)*

**Registers at call:** **Return Registers:**
SI = 000Bh *unknown*
ES:BX -> parameters (see Table 22-7)
**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Function 0003h

**22-7 Format of DMA buffer parameters:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to memory block to contain DMA buffer |
| 04h | WORD | desired size of DMA buffer |

## Media Vision

FM.COM is an Ad Lib SOUND.COM-compatible driver for Media Vision's Pro Audio Spectrum sound boards. Media Vision's PCM.COM is a superset of the standard PCM driver which provides additional functions for fine control of the driver; see the "PCM driver" section above for the standard PCM driver calls. MVSOUND.SYS is a driver for the Media Vision ProAudio Spectrum family of sound boards whose primary programmer was Bryan Crane.

---

### INT 2Fh Function BC00h

*Media Vision MVSOUND.SYS*

### *INSTALLATION CHECK*

**Registers at call:**
AX = BC00h
BX = 3F3Fh ('??')
CX = 0000h
DX = 0000h

**Return Registers:**
if installed, BX XOR CX XOR DX = 4D56h ('MV')

**Conflicts:** EGA.SYS (Chapter 43); see also Table 2-2
**See also:** Functions BC01h to BC04h and BC06h

---

### INT 2Fh Function BC01h

*Media Vision MVSOUND.SYS*

### *GET VERSION*

**Registers at call:**
AX = BC00h
BX = 6D20h ('m ')
CX = 2076h (' v')
DX = 2020h ('  ')

**Return Registers:**
BX = ASCII major version
CX = ASCII minor version (leading zeros significant)

**Conflicts:** EGA.SYS (Chapter 43); see also Table 2-2
**See also:** Function BC00h

---

### INT 2Fh Function BC02h

*Media Vision MVSOUND.SYS*

### *GET STATE TABLE POINTER*

**Registers at call:**
AX = BC02h

**Return Registers:**
BX:DX -> state table

**Conflicts:** EGA.SYS (Chapter 43); see also Table 2-2
**See also:** Functions BC00h and BC03h

---

### INT 2Fh Function BC03h

*Media Vision MVSOUND.SYS*

### *GET FUNCTION TABLE POINTER*

**Registers at call:**
AX = BC03h

**Return Registers:**
BX:DX -> function table

**Conflicts:** EGA.SYS (Chapter 43); see also Table 2-2
**See also:** Functions BC00h and BC02h

---

### INT 2Fh Function BC04h

*Media Vision MVSOUND.SYS*

### *GET DMA AND IRQ CHANNELS*

**Registers at call:**
AX = BC04h

**Return Registers:**
AX = 4D56h ('MV')
BL = DMA channel
CL = IRQ number

**Conflicts:** EGA.SYS (Chapter 43); see also Table 2-2
**See also:** Functions BC00h and BC01h

---

### INT 2Fh Function BC06h

*Media Vision MVSOUND.SYS*

### *GET STATUS STRING*

**Registers at call:**
AX = BC06h

**Return Registers:**
AX = 4D56h ('MV')
DX:BX -> status string

**Details:** The first byte of the status string is 0Ch if there no status message to display.
**Conflicts:** EGA.SYS (Chapter 43); see also Table 2-2
**See also:** Functions BC00h, BC01h, and BC0Bh

## INT 2Fh Function BC00h

---

**INT 2Fh Function BC0Bh** *Media Vision MVSOUND.SYS*

**Registers at call:**
AX = BC0Bh

**Return Registers:**
BX:DX -> ASCIZ path for MVPROAS.EXE,
or 0000h:0000h if not available

**Conflicts:** EGA.SYS (Chapter 43); see also Table 2-2
**See also:** Function BC00h

---

**INT 65h Function 8000h** *Media Vision FM.COM v4.1a+*
***GET INTERNAL DATA STRUCTURES*** *(partially doc.)*

**Registers at call:**
SI = 8000h

**Return Registers:**
DX:AX -> internal data structures

**Conflicts:** see Table 2-3
**See also:** Function 8001h

---

**INT 65h Function 8001h** *Media Vision FM.COM v4.1a+*
***GET VOICE COUNT*** *(partially doc.)*

**Registers at call:**
SI = 8001h

**Return Registers:**
AX = *unknown*
DX = *number of voices* (09h or 0Bh)

**Conflicts:** see Table 2-3
**See also:** Function 8000h

---

**INT 65h Function 8002h** *Media Vision FM.COM v4.1a+*
***START BACKGROUND FM SOUNDS***

**Registers at call:**
SI = 8002h

**Return Registers:**
none

**Conflicts:** see Table 2-3
**See also:** Function 8003h

---

**INT 65h Function 8003h** *Media Vision FM.COM v4.1a+*
***STOP BACKGROUND FM SOUNDS***

**Registers at call:**
SI = 8003h

**Return Registers:**
none

**Conflicts:** see Table 2-3
**See also:** Function 8002h

---

**INT 65h Function 8004h** *Media Vision FM.COM v4.1a+*
***GET UNKNOWN VALUES*** *(undocumented)*

**Registers at call:**
SI = 8004h

**Return Registers:**
AX = *unknown* (0280h)
DX = *unknown* (01A0h)

**Conflicts:** see Table 2-3

---

**INT 65h Function 8005h** *Media Vision FM.COM v4.1a+*
***UNDETERMINED FUNCTION*** *(undocumented)*

**Registers at call:**
SI = 8005h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Conflicts:** see Table 2-3
**See also:** Function 8000h

### INT 94h Function 000Dh
### *GET STATUS*

*Media Vision PCM.COM (partially doc.)*

**Registers at call:**
SI = 000Dh

**Return Registers:**
AX = status (0000h = waiting) (see Table 22-8)

**Conflicts:** IBM ROM BASIC (Chapter 54)

**22-8 Bitfields for PCM.COM status:**

| | |
|---|---|
| bit 0 | playing |
| bit 1 | recording |
| bit 2 | SBplaying |
| bit 3 | SBrecording |
| bit 14 | SBpaused |
| bit 15 | paused |

### INT 94h Function 8000h
### *GET INTERNAL DMA BUFFER ADDRESS*

*Media Vision PCM.COM (partially doc.)*

**Registers at call:**
SI = 8000h

**Return Registers:**
DX:AX -> DMA buffer

**Details:** The installation check for the Media Vision PCM.COM "shark" functions consists of testing for the signature "PCM-SHARK" at offset 107h in the INT 94h handler's segment.

**Conflicts:** IBM ROM BASIC (Chapter 54)

**See also:** Functions 8001h and 8004h

### INT 94h;Function 8001h
### *GET INTERNAL DMA BUFFER SIZE AND DIVISIONS*

*Media Vision PCM.COM (partially doc.)*

**Registers at call:**
SI = 8001h

**Return Registers:**
AX = DMA buffer size
DX = divisions

**Conflicts:** IBM ROM BASIC (Chapter 54)

**See also:** Function 8000h

### INT 94h Function 8002h
### *CHECK BOARD ADDRESS*

*Media Vision PCM.COM (partially doc.)*

**Registers at call:**
SI = 8002h

**Return Registers:**
AX = status
 0000h if board not at specified I/O address
 other if board found

**Details:** The I/O address is specified by ORing the base I/O port shifted left four bits into SI before calling INT 94h.

**Conflicts:** IBM ROM BASIC (Chapter 54)

**See also:** Function 8000h

### INT 94h Function 8004h
### *GET INTERNAL NOTE BUFFER*

*Media Vision PCM.COM (partially doc.)*

**Registers at call:**
SI = 8004h

**Return Registers:**
AX = offset of note buffer (segment = segment of internal DMA buffer)
DX = size of buffer in note structures

**Conflicts:** IBM ROM BASIC (Chapter 54)

**See also:** Function 8000h

---

**INT 94h Function 8005h** *Media Vision PCM.COM*
***SINGLE-STEP QUEUE*** *(partially doc.)*

**Registers at call:**
SI = 8005h

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)

---

**INT 94h Function 8011h** *Media Vision PCM.COM*
***INITIALIZE*** *(partially doc.)*

**Registers at call:**
SI = 8011h
ES:BX -> "iobf91" structure

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)

---

**INT 94h Function 8012h** *Media Vision PCM.COM*
***LOAD SOUND FOR LATER PLAY THROUGH KEYBOARD*** *(partially doc.)*

**Registers at call:**
SI = 8012h
ES:BX -> "i94f92buf" structure

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Functions 8013h and 8014h

---

**INT 94h Function 8013h** *Media Vision PCM.COM*
***GET INTERNAL SOUND USAGE*** *(partially doc.)*

**Registers at call:**
SI = 8013h

**Return Registers:**
AX = number of sounds used
DX = maximum handles

**Conflicts:** IBM ROM BASIC (Chapter 54)

---

**INT 94h Function 8014h** *Media Vision PCM.COM*
***GET DATA FOR SPECIFIED SOUND*** *(partially doc.)*

**Registers at call:**
SI = 8014h
ES:BX -> "i94f92buf" structure to be filled in (sound number field set to desired sound)

**Return Registers:**
AX = status
  0000h successful
  FFFFh sound number out of range

**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Functions 8012h and 8013h

---

**INT 94h Function 8015h** *Media Vision PCM.COM*
***GET/SET INTERNAL DMA BUFFER*** *(partially doc.)*

**Registers at call:**
SI = 8015h
ES:BX -> DMA info structure (see Table 22-9)

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)

**22-9 Format of DMA info structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to DMA buffer (offset FFFFh = return current buffer info) |
| 04h | WORD | DMA buffer size |
| 06h | WORD | divisions |

---

### INT 94h Function 8016h
### *SIMULATE DOUBLE-SHIFT HOTKEY*

*Media Vision PCM.COM (partially doc.)*

**Registers at call:**
SI = 8016h
AX = hotkey number (01h-08h)

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Function 8017h Subfunction 02h

---

### INT 94h Function 8017h, Subfunction 01h
### *CTRL-G INTERCEPT*

*Media Vision PCM.COM (partially doc.)*

**Registers at call:**
SI = 8017h
AL = 01h
AH = new state (00h off, 01h on)

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Function 8017h Subfunction 02h

---

### INT 94h Function 8017h, Subfunction 02h
### *DOUBLE-SHIFT-HOTKEY SOUND FEATURE*

*Media Vision PCM.COM (partially doc.)*

**Registers at call:**
SI = 8017h
AL = 02h
AH = new state (00h off, 01h on)

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Function 8017h Subfunction 04h

---

### INT 94h Function 8017h, Subfunction 04h
### *RANDOM SOUND FEATURE*

*Media Vision PCM.COM (partially doc.)*

**Registers at call:**
AL = 04h
SI = 8017h
AH = new state
- 00h off
- 01h on
  - CX = minimum delay
  - DX = maximum delay

**Return Registers:**
*unknown*

**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Function 8017h Subfunction 02h

---

### INT 94h Function 8017h, Subfunction 08h
### *NO ACTIVITY FEATURE*

*Media Vision PCM.COM (partially doc.)*

**Registers at call:**
AL = 08h
SI = 8017h
AH = new state
- 00h off
- 01h on
  - DX:CX = delay

**Return Registers:**
*unknown*

**See also:** Function 8017h Subfunction 10h
**Conflicts:** IBM ROM BASIC (Chapter 54)

---

### INT 94h Function 8017h, Subfunction 10h
*Media Vision PCM.COM (partially doc.)*

**TIMER CONTROL**

| Registers at call: | Return Registers: |
|---|---|
| AL = 10h | *unknown* |
| SI = 8017h | |

AH bit 7: set timer
AH bit 6: timer active (timer turned off if clear)
AH bits 5-0: timer number
DX:CX = delay if AH bit 7 set (one-shot if DX bit 15 set)

**Conflicts:** IBM ROM BASIC (Chapter 54)
**See also:** Function 8017h Subfunction 08h

---

### INT 94h Function 8018h
*Media Vision PCM.COM (partially doc.)*

**GET INFO**

| Registers at call: | Return Registers: |
|---|---|
| SI = 8018h | DX:AX -> desired information |

AL = what to get
  00h "F92state"
  01h "F92bkgd"
  02h "I10timer"
  03h "I08state"

**Conflicts:** IBM ROM BASIC (Chapter 54)

## Tandy

This section describes API features for the unique sound capabilities included in certain Tandy computer systems.

---

### INT 1Ah Function 7Fh
*Tandy 2500, Tandy 1000L series*

**DIGITAL SOUND**

| Registers at call: | Return Registers: |
|---|---|
| AH = 7Fh | *unknown* |
| *others, if any, unknown* | |

**See also:** Functions 80h, 83h, and 85h

---

### INT 1Ah Function 80h
*PCjr, Tandy*

**SET UP SOUND MULTIPLEXOR**

| Registers at call: | Return Registers: |
|---|---|
| AH = 80h | none |

AL = source

| | |
|---|---|
| 00h | 8253 channel 2 |
| 01h | cassette input |
| 02h | I/O channel "Audio IN" |
| 03h | sound generator chip |

**Conflicts:** PCMCIA Socket Services (Chapter 5)
**See also:** Functions 7Fh and 83h

---

### INT 1Ah Function 8100h
*Tandy 2500, Tandy 1000L series*

**DIGITAL SOUND - INSTALLATION CHECK**

| Registers at call: | Return Registers: |
|---|---|
| AX = 8100h | AH > 80h if supported |

**Conflicts:** PCMCIA Socket Services (Chapter 5)
**See also:** Functios 83h and 84h

### INT 1Ah Function 83h
*Tandy 2500, Tandy 1000L series*

***START PLAYING DIGITAL SOUND***

**Registers at call:**
AH = 83h
AL = volume (0=lowest, 7=highest)
CX = number of bytes to play
DX = time between sound samples (multiples of 273 nanoseconds)
ES:BX -> sound data (array of 8-bit PCM samples)

**Return Registers:**
*unknown*

**Details:** This call returns immediately while the sound plays in the background.

The sound data pointed to by ES:BX probably shouldn't cross a 64k boundary. The BIOS appears to call INT 15h Function 91FBh when the sound device underflows to allow another INT 1Ah Function 83h for seamless playing of long sounds.

**Conflicts:** PCMCIA Socket Services (Chapter 5)

**See also:** Functions 8100h and 84h, INT 15h Function 91h (Chapter 4)

### INT 1Ah Function 84h
*Tandy 2500, Tandy 1000L series*

***STOP PLAYING DIGITAL SOUND***

**Registers at call:**
AH = 84h

**Return Registers:**
*unknown*

**Conflicts:** PCMCIA Socket Services (Chapter 5)

**See also:** Functions 8100h, 83h and 85h

### INT 1Ah Function 85h
*Tandy 2500, Tandy 1000L series*

***DIGITAL SOUND***

**Registers at call:**
AH = 85h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Conflicts:** PCMCIA Socket Services (Chapter 5)

**See also:** Functions 7Fh and 83h

## SND

The SND API is a simple interface for generating single tones and strings of notes. It is supported by at least IC (Internal Commands) v2.0, a shareware TSR by Geoff Friesen which extends COMMAND.COM's internal command set.

### INT 1Ah Function FF00h
*SND*

***INSTALLATION CHECK***

**Registers at call:**
AX = FF00h

**Return Registers:**
AL = *version* (02h)
AH = busy flag (00h if not in a SND call, 01h if SND currently active)

**Conflicts:** AT&T 6300 (Chapter 7), RighTime (Chapter 66)

**See also:** Functions FF01h, FF02h, FF04h, and FF05h

### INT 1Ah Function FF01h
*SND*

***PAUSE***

**Registers at call:**
AX = FF01h
DX = number of clock ticks to delay

**Return Registers:**
AH = status
- 00h successful
- 01h SND busy

**Details:** If successful, execution returns to the caller after the delay expires; if SND is busy, execution returns immediately. The IC v2.0 implementation of this API makes no special allowance for time rollover at midnight, which can cause the delay to be over one hour if this function is called just before the BIOS time count rolls over and the delay extends into the next day.

**Conflicts:** AT&T 6300 (Chapter 7), RighTime (Chapter 66)
**See also:** Function FF00h, SoundBlaster INT 80h Function 0009h (above), BIOS INT 15h Function 86h (Chapter 4), DR Multiuser DOS INT E0h Function BDh (Chapter 48), Fast-Graph/Light INT 62h Function 0096h (Chapter 52), INT 7Fh Function E8h (NI)

---

### INT 1Ah Function FF02h *SND*
### *START SOUND*

**Registers at call:**
AX = FF02h
DX = frequency in Hertz (14h-FFFFh)

**Return Registers:**
AH = status
00h successful
01h SND busy

**Conflicts:** AT&T 6300 (Chapter 7), RighTime (Chapter 66)
**See also:** Functions FF00h, FF01h, and FF03h

---

### INT 1Ah Function FF03h *SND*
### *STOP SOUND*

**Registers at call:**
AX = FF03h

**Return Registers:**
AH = status
00h successful
01h busy

**Details:** This function turns off any sound currently being emitted by the PC's speaker unless SND is currently busy processing an API call (this includes background music). Use Function FF05h to stop the sound even if an API call is in progress.
**Conflicts:** AT&T 6300 (Chapter 7), RighTime (Chapter 66)
**See also:** Functions FF00h, FF02h, and FF05h

---

### INT 1Ah Function FF04h *SND*
### *PLAY MUSIC STRING IN BACKGROUND*

**Registers at call:**
AX = FF04h
DS:DX -> ASCIZ music string

**Return Registers:**
AH = status
00h successful (music begins playing in backgroun
01h busy

**Details:** The music string accepted by SND is not the same as that accepted by BASIC and other programs which process music strings.
**Conflicts:** AT&T 6300 (Chapter 7), RighTime (Chapter 66)
**See also:** Functions FF00h and FF05h, SoundBlaster INT 80h Function 0006h (above)

---

### INT 1Ah Function FF05h *SND*
### *UNCONDITIONALLY STOP SOUND*

**Registers at call:**
AX = FF05h

**Return Registers:**
AH = 00h (successful)

**Details:** This function is the same as Function FF03h, but will stop the sound even if SND is currently busy, such as playing background music.
**Conflicts:** AT&T 6300 (Chapter 7), RighTime (Chapter 66)
**See also:** Functions FF00h and FF03h, SoundBlaster INT 80h Function 0007h (above)

## RESPLAY

RESPLAY is a freeware sound sampling/playback utility by Mark J. Cox.

---

### INT 2Fh Function 8200h *RESPLAY*
### *SAMPLE/PLAYBACK*

**Registers at call:**
AX = 8200h

DX:DI -> start of sample space
CX:BX = length in bytes

**Return Registers:**
AX = status
1000h successful
2000h not initialized (see Function 8210h)
other RESPLAY not installed

**Conflicts:** Nanosoft CAPDOS (NI); see also Table 2-2
**See also:** Functions 8201h and 8210h

---

### INT 2Fh Function 8201h
*RESPLAY*
### *INSTALLATION CHECK*

**Registers at call:**
AX = 8201h

**Return Registers:**
AX = 7746h if installed

**Conflicts:** Nanosoft CAPDOS (NI); see also Table 2-2
**See also:** Function 8202h

---

### INT 2Fh Function 8202h
*RESPLAY*
### *UNINSTALL*

**Registers at call:**
AX = 8202h

**Return Registers:**
AX = status
1000h successful

**Conflicts:** Nanosoft CAPDOS (NI); see also Table 2-2
**See also:** Function 8201h

---

### INT 2Fh Function 8210h
*RESPLAY*
### *INITIALIZE*

**Registers at call:**
AX = 8210h
BL = sound device
- 00h printer port LPT1
- 01h printer port LPT2
- 02h prototype board at I/O address 0300h
- 03h printer port (alternative LPT1)
- 04h internal speaker

BH = sample rate in multiples of 250 Hz (14h to A0h)
CL = direction
- 00h playback
- 01h sample

**Return Registers:**
AX = status
1000h successful
2000h parameter out of range
other RESPLAY not installed

**Conflicts:** Nanosoft CAPDOS (NI); see also Table 2-2
**See also:** Function 8200h

## IBMSND driver

The IBMSND driver is part of John W. Ratcliff's The IBM Digitized Sound Package.

---

### INT 66h Function 0688h
*IBMSND driver*
### *PLAY 8-BIT DIGITIZED SOUND*

**Registers at call:**
AX = 0688h
DS:SI -> SNDSTRUC (see Table 22-10)

**Return Registers:**
*unknown*

**Details:** The installation check consists of looking for a valid signature string six bytes prior to the interrupt handler; this string may be either "KERN" or "MIDI" (in the latter case, call Function 0701h to determine whether IBMSND is installed).
**Conflicts:** see Table 2-3
**See also:** Functions 068Bh, 068Fh, and 0701h

**22-10 Format of SNDSTRUC:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to audio data |
| 04h | WORD | length of audio data in bytes |
| 06h | DWORD | pointer to playback status flag |
| 0Ah | WORD | playback frequency |

---

### INT 66h Function 0689h
*IBMSND driver*

### *REPORT SOUND DRIVER STATUS*

**Registers at call:**
AX = 0689h

**Return Registers:**
AX = status
0000h no sound playing
0001h sound effect is currently playing

**Conflicts:** see Table 2-3
**See also:** Functions 0688h, 068Bh, and 068Ch

---

### INT 66h Function 068Ah
*IBMSND driver*

### *PREFORMAT SOUND*

**Purpose:** Convert audio data into output hardware format.

**Registers at call:**
AX = 068Ah
DS:SI -> SNDSTRUC (see Table 22-10)

**Return Registers:**
none

**Conflicts:** see Table 2-3
**See also:** Function 068Bh

---

### INT 66h Function 068Bh
*IBMSND driver*

### *PLAY PREFORMATTED SOUND*

**Registers at call:**
AX = 068Bh
DS:SI -> SNDSTRUC (see Table 22-10)

**Return Registers:**
AX = *unknown*

**Conflicts:** see Table 2-3
**See also:** Functions 0688h, 068Ah, and 068Fh

---

### INT 66h Function 068Ch
*IBMSND driver*

### *REPORT AUDIO DRIVER CAPABILITIES*

**Registers at call:**
AX = 068Ch

**Return Registers:**
AX = capabilities (see Table 22-11)
DX = playback rate if fixed-frequency playback

**Conflicts:** see Table 2-3
**See also:** Functions 0689h and 068Dh

**22-11 Bitfields for capabilities:**

| | |
|---|---|
| bit 0 | can play audio in background |
| bit 1 | data is massaged |
| bit 2 | driver plays at fixed frequency, resampling input data to fit |
| bit 3 | driver uses timer interrupt |

---

### INT 66h Function 068Dh
*IBMSND driver*

### *REPORT CURRENT SAMPLE ADDRESS*

**Purpose:** Determine what point in the audio data the playback has reached, for synchronization with video or animation effects.

**Registers at call:**
AX = 068Dh

**Return Registers:**
AX = current playback address

**Details:** This function applies to background playback only. The reported address may be an approximation rather than the exact address.
**Conflicts:** see Table 2-3
**See also:** Functions 068Ch and 0691h

---

### INT 66h Function 068Eh *IBMSND driver*
### *SET CALLBACK ADDRESS*

**Purpose:** Specify the function to be called when playback of a sound effect is completed.

**Registers at call:**
AX = 068Eh
BX:DX -> callback function
(0000h:0000h to disable callback)
DS = value to load into DS when calling the callback function

**Return Registers:**
none

**Details:** The callback function will typically be called during a hardware interrupt, so all the usual precautions should be taken except for preserving registers.
**Conflicts:** see Table 2-3
**See also:** Function 0691h

---

### INT 66h Function 068Fh *IBMSND driver*
### *STOP CURRENT SOUND*

**Purpose:** Cause any currently-playing sound effect to be terminated.

**Registers at call:**
AX = 068Fh

**Return Registers:**
none

**Conflicts:** see Table 2-3
**See also:** Functions 0688h and 068Bh

---

### INT 66h Function 0690h *IBMSND driver*
### *"SetAudioHardware" SET UP HARDWARE INFO* *[obsolete]*

**Registers at call:**
AX = 0690h

**Return Registers:**
none

**Conflicts:** see Table 2-3
**Details:** This function is no longer implemented.

---

### INT 66h Function 0691h *IBMSND driver*
### *REPORT CALLBACK ADDRESS*

**Registers at call:**
AX = 0691h

**Return Registers:**
AX:DX -> current callback function
BX = original caller's DS register

**Conflicts:** see Table 2-3
**See also:** Function 068Eh

---

### INT 66h Function 0701h *IBM Digitized Sound Package MIDI driver*
### *GET DIGITIZED SOUND CAPABILITIES*

**Registers at call:**
AX = 0701h

**Return Registers:**
AX = digitized sound capabilities
0000h if digitized sound driver (functions 06xxh) not available

**Details:** The installation check for the MIDI driver is to test for the signature "MIDI" six bytes before the interrupt handler.
**Conflicts:** see Table 2-3
**See also:** Function 0688h

## FARTBELL

FARTBELL is a joke program by Guenther Thiele which makes various noises when programs output a bell.

---

### INT 21h Function 4000h

*FARTBELL.EXE*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = 4000h | CF clear if installed |
| BX = 0002h (handle for STDERR) | AX = CS of resident code |
| CX = 0000h | |
| DS:DX = 0000h:0000h | |

**Conflicts:** MS-DOS (Chapter 24)

**See also:** Function 4001h

---

### INT 21h Function 4001h

*FARTBELL.EXE*

***FORCE NOISE***

| Registers at call: | Return Registers: |
|---|---|
| AX = 4001h | none |
| BX = 0002h (handle for STDERR) | |
| CX = 0000h | |
| DS:DX = 0000h:0000h | |

**See also:** Function 4000h

## Speech

One SPEECH.COM is a resident text-to-speech converter by Douglas Sisco, the other is by Andy C. McGuire.

---

### INT F1h

*Douglas Sisco SPEECH.COM*

***CONVERT TEXT STRING TO SPEECH***

| Registers at call: | Return Registers: |
|---|---|
| DS:BX -> '$'-terminated text string | none |

---

### INT F1h

*Andy C. McGuire SPEECH.COM/SAY.COM*

***API***

**See also:** INT F2h

---

### INT F2h

*Andy C. McGuire SPEECH.COM/SAY.COM*

***API***

**See also:** INT F1h

## CHAPTER 23

# *Screen Readers and Magnifiers*

The computer is transformed into a valuable tool for many visually-impaired people through the addition of screen reading and screen magnifying programs. A screen reader extracts text from the screen and sends it to a speech synthesizer, while a screen magnifier displays a portion of the screen in large-sized text filling the entire screen.

## AccessDOS

AccessDOS is a public domain TSR developed at The Trace Research and Development Center which provides extensions for keyboard, mouse, and sound access by the visually, hearing, or motor-control impaired. It has been made available on the MS-DOS 6.0 Supplemental Disk.

INT 60h is the default vector for AccessDOS; AccessDOS will use the first 0000h:0000h vector in the range 60h through 66h. As with all TSRs that use interrupts in this range, there are many potential conflicts, listed in Table 2-3.

---

### INT 60h Function ADh

*AccessDOS*

***API***

**Registers at call:**
AH = ADh
AL = API function
    E1h *unknown*
    E2h get configuration

**Return Registers:**
AX = *unknown*
BX:AX -> configuration data

**Conflicts:** see Table 2-3

## ASAP

ASAP (Automatic Screen Access Program) is a shareware screen reader by MicroTalk.

---

### INT 14h Function F0F0h

*ASAP v1.0*

***UNDETERMINED FUNCTION***

**Registers at call:**
AX = F0F0h
DX = *unknown*
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function F0F1h

---

### INT 14h Function F0F1h

*ASAP v1.0*

***INSTALLATION CHECK***

**Registers at call:**
AX = F0F1h
DX = 0000h

**Return Registers:**
DX = segment of resident code, 0000h if not installed

**See also:** Function F0F0h, Tinytalk INT 10h Function 3800h (below)

## AutoBraille

AutoBraille is a shareware text-to-braille converter by KANSYS, Inc.

---

**INT 2Fh Function FB00h**
***INSTALLATION CHECK***
*AutoBraille v1.1A (undocumented)*

**Registers at call:**
AX = FB00h

**Return Registers:**
AX = 00FFh if installed

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2
**See also:** ASAP INT 14h Function F0F1h (above), Tinytalk INT 10h Function 3800h (below)

---

**INT 2Fh Function FB01h**
***UNDETERMINED FUNCTION***
*AutoBraille v1.1A (undocumented)*

**Registers at call:**
AX = FB01h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2

---

**INT 2Fh Function FB02h**
***UNDETERMINED FUNCTION***
*AutoBraille v1.1A (undocumented)*

**Registers at call:**
AX = FB02h

**Return Registers:**
AH = *unknown*
AL = *unknown*

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2

---

**INT 2Fh Function FB03h**
***GET NEXT UNKNOWN***
*AutoBraille v1.1A (undocumented)*

**Registers at call:**
AX = FB03h

**Return Registers:**
AX = *unknown*

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2

---

**INT 2Fh Functions FB04h to FB08h**
***SET UNKNOWN VALUE OR FLAG***
*AutoBraille v1.1A (undocumented)*

**Registers at call:**
AH = FBh
AL = 04h-08h

**Return Registers:**
AX = 0000h

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2

---

**INT 2Fh Functions FB09h to FB0Fh**
***SET UNKNOWN VALUE OR FLAG***
*AutoBraille v1.1A (undocumented)*

**Registers at call:**
AH = FBh
AL = 09h-0Fh (0Eh = COM1,
0Fh = COM2)

**Return Registers:**
*unknown*

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2

### INT 2Fh Functions FB10h to FB1Fh

*AutoBraille v1.1A (undocumented)*

***SET UNKNOWN VALUE OR FLAG***

**Registers at call:**
AH = FBh
AL = 10h-1Fh
*others, if any, unknown*

**Return Registers:**
*unknown*

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2

### INT 2Fh Function FB20h

*AutoBraille v1.1A (undocumented)*

***SET UNKNOWN FLAGS***

**Registers at call:**
AX = FB20h
BL = flags to set

**Return Registers:**
none

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2

**See also:** Function FB21h

### INT 2Fh Function FB21h

*AutoBraille v1.1A (undocumented)*

***CLEAR UNKNOWN FLAGS***

**Registers at call:**
AX = FB21h
BL = flags to clear

**Return Registers:**
none

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2

**See also:** Function FB20h

### INT 2Fh Function FB22h

*AutoBraille v1.1A (undocumented)*

***SET UNKNOWN VALUE***

**Registers at call:**
AX = FB22h
BL = *unknown*

**Return Registers:**
none

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2

### INT 2Fh Functions FB28h and FB29h

*AutoBraille v1.1A (undocumented)*

***UNDETERMINED FUNCTIONS***

**Registers at call:**
AX = FB28h and FB29h
BX = *unknown*
*others, if any, unknown*

**Return Registers:**
*unknown*

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2

**See also:** Function FB29h

### INT 2Fh Functions FB2Bh to FB34h

*AutoBraille v1.1A (undocumented)*

***SET UNKNOWN VALUE***

**Registers at call:**
AH = FBh
AL = 2Bh-34h
BX = *unknown*

**Return Registers:**
none

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2

---

### INT 2Fh Functions FB35h to FB37h
### *SET UNKNOWN VALUES*

*AutoBraille v1.1A (undocumented)*

**Registers at call:**
AX = FB35h
BL = *unknown*

**Return Registers:**
none

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2

---

### INT 2Fh Function FB64h
### *GET UNKNOWN VALUE*

*AutoBraille v1.1A (undocumented)*

**Registers at call:**
AX = FB64h

**Return Registers:**
AX = *unknown* (0006h seen)

**Conflicts:** SoundBlaster (Chapter 22), Borland DOS Extenders (Chapter 36), Jot-It (Chapter 66); see also Table 2-2

## MAGic

MAGic (MAGnification In Color) is a TSR by Microsystems Software, Inc. providing 2x2 text and graphics magnification on VGA, XGA, and SVGA.

INT 49h is the default vector for MAGic, but may be overridden on the commandline. The actual interrupt in use may be found by searching for the signature "MAGic" or "xMAGic" (for the deluxe version) immediately preceding the interrupt handler (this is also the installation check). MAGic uses CodeRunneR, which places the signature "RT" at offset 0000h in the interrupt handler's segment, followed by MAGic's TSR ID of "VMAG".

---

### INT 49h Function 0001h
### *TURN ON MAGNIFICATION*

*MAGic v1.16+*

**Registers at call:**
AX = 0001h

**Return Registers:**
AX = status (see Table 23-1)
BX,CX,DX destroyed

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Texas Instruments PC (Chapter 8), PCjr (Chapter 13), Watstar PC Network (NI)

**See also:** Functions 0002h, 0003h, 0004h, and 0008h

**23-1 MAGic status codes:**

| | |
|---|---|
| 0000h | cannot magnify current video mode |
| 0002h | magnified (text mode) |
| 0003h | magnified (graphics mode) |
| FFFDh | function works only in magnified mode |
| FFFFh | MAGic busy, retry later |

---

### INT 49h Function 0002h
### *TURN OFF MAGNIFICATION*

*MAGic v1.16+*

**Registers at call:**
AX = 0002h

**Return Registers:**
AX = status (see Table 23-1)
BX,CX,DX destroyed

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Texas Instruments PC (Chapter 8), PCjr (Chapter 13), Watstar PC Network (NI)

**See also:** Function 0001h

---

## INT 49h Function 0003h

*MAGic v1.16+*

### *SHIFT MAGNIFIED WINDOW TO INCLUDE SPECIFIED LOCATION*

| Registers at call: | Return Registers: |
|---|---|
| AX = 0003h | AX = status (0000h,FFFFh) (see Table 23-1) |
| BX = vertical position (character row [text] or pixel row [graphics]) | BX,CX,DX destroyed |
| DX = horizontal position (char column [text] or 8-pixel units [gr]) | |

**Details:** The window is not moved if the specified position is already inside the current window.

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Texas Instruments PC (Chapter 8), PCjr (Chapter 13), Watstar PC Network (NI)

**See also:** Functions 0001h, 0004h, and 0005h

---

## INT 49h Function 0004h

*MAGic v1.16+*

### *REPOSITION MAGNIFIED WINDOW*

| Registers at call: | Return Registers: |
|---|---|
| AX = 0004h | AX = status (0000h,FFFFh) (see Table 23-1) |
| BX = vertical position of upper left corner | BX,CX,DX destroyed |
| DX = horizontal position | |

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Texas Instruments PC (Chapter 8), PCjr (Chapter 13), Watstar PC Network (NI)

**See also:** Functions 0001h, 0003h, and 0005h

---

## INT 49h Function 0005h

*MAGic v1.16+*

### *GET POSITION OF MAGNIFIED WINDOW*

**Registers at call:**

AX = 0005h

**Return Registers:**

CX destroyed

AX = status

- 0000h successful
  - BX = vertical position (char row or pixel row)
  - DX = horizontal position (char column or 8-pixel units)
- FFFFh MAGic busy, retry later
  - BX,DX destroyed

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Texas Instruments PC (Chapter 8), PCjr (Chapter 13), Watstar PC Network (NI)

**See also:** Functions 0001h, 0003h, 0004h, 0006h, and 0007h

---

## INT 49h Function 0006h

*MAGic v1.16+*

### *GET SIZE OF FULL SCREEN*

**Registers at call:**

AX = 0006h

**Return Registers:**

AX = status

- 0000h successful
  - BX = vertical size (char rows or pixel rows)
  - DX = horizontal size (char cols or 8-pixel units
- FFFFh MAGic busy, retry later
  - BX,DX destroyed

CX destroyed

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Texas Instruments PC (Chapter 8), PCjr (Chapter 13), Watstar PC Network (NI)

**See also:** Functions 0001h, 0005h, and 0007h

---

### INT 49h Function 0007h

*MAGic v1.16+*

### *GET SIZE OF MAGNIFICATION WINDOW*

**Registers at call:**
AX = 0007h

**Return Registers:**
AX = status
  0000h successful
    BX = vertical size (char rows or pixel rows)
    DX = horizontal size (char cols or 8-pixel units)
  FFFEh invalid function
  FFFFh MAGic busy, retry later
    BX,DX destroyed
CX destroyed

**BUG:** In versions 1.16 and 1.17, this function is not recognized as valid, but AX=0000h is accepted and will branch to a random location.
**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Texas Instruments PC (Chapter 8), PCjr (Chapter 13), Watstar PC Network (NI)
**See also:** Functions 0001h and 0006h

---

### INT 49h Function 0008h

*MAGic v1.23+*

### *SET TEXT MODE MAGNIFICATION SIZE*

**Registers at call:**
AX = 0008h
BX = scaling factor (01h=1.4 times,
  02h, 04h, 06h, 08h, 09h=12 times)

**Return Registers:**
AX = status
  0000h successful
  FFFBh scaling factor only available in MAGic Deluxe
  FFFCh already in magnified state, can't set size

**Details:** This call specifies the amount by which a subsequent call to Function 0001h should magnify the display.

Scaling factors greater than 2 are only available in MAGic Deluxe.

**Conflicts:** Z100 (Chapter 7), Acorn BBC Master (Chapter 7), Texas Instruments PC (Chapter 8), PCjr (Chapter 13), Watstar PC Network (NI)
**See also:** Function 0001h

## Tinytalk Personal

Tinytalk is a shareware screen reader by OMS Development/Eric Bohlman.

---

### INT 10h Function 3800h

*Tinytalk Personal v1.09f+*

### *GET CONFIGURATION INFO*

**Registers at call:**
AX = 3800h

**Return Registers:**
ES:DI -> configuration info (see Table 23-2)

**Details:** This call is also used as the installation check by verifying that the returned ES:DI points at valid configuration information.

Tinytalk v1.09f ignores AL and always returns the configuration info; v1.10 adds several subfunctions selected with AL.

**See also:** Functions 3801h, 3803h, 3806h, and 39h; INT 14h Function F0F1h; AutoBraille INT 2Fh/FB00h

**23-2 Format of Tinytalk configuration info:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 8 BYTEs | signature "TTCONFIG" |
| 08h | WORD | size of configuration data, not counting signature, this WORD, or the following byte |
| 0Ah | BYTE | *unknown* |
| 0Bh | ? | configuration data |

---

**INT 10h Functions 3801h and 3802h** *Tinytalk Personal v1.10*

***UNDETERMINED FUNCTIONS***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 3801h and 3802h | *unknown* |
| DL = *unknown* | |

**See also:** Function 3800h

---

**INT 10h Function 3803h** *Tinytalk Personal v1.10*

***GET UNKNOWN VALUE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 3803h | AL = *unknown* |

**See also:** Functions 3800h, 3804h, and 3805h

---

**INT 10h Function 3804h** *Tinytalk Personal v1.10*

***GET UNKNOWN VALUE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 3804h | AX = *unknown* |

**See also:** Functions 3800h, 3803h, and 3805h

---

**INT 10h Function 3805h** *Tinytalk Personal v1.10*

***GET UNKNOWN VALUE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 3805h | AL = *unknown* |

**See also:** Functions 3800h, 3803h, and 3804h

---

**INT 10h Function 3806h** *Tinytalk Personal v1.10*

***UNDETERMINED FUNCTION***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 3806h | *unknown* |
| ES:DX -> ASCIZ *unknown* | |

**See also:** Function 3800h

---

**INT 10h Function 39h** *Tinytalk Personal v1.10*

***UNDETERMINED FUNCTION***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 39h | none |

**Details:** This function was a NOP in TTDEMO.EXE, but may be implemented in other variants of Tinytalk.

## CHAPTER 24

# *MS-DOS Kernel*

This chapter tabulates the functions provided by the MS-DOS (and compatible) kernel, which consists of the two hidden system files variously called IO.SYS/MSDOS.SYS or IBMBIO.COM/IBMDOS.COM. Since most of these routines have been widely documented elsewhere, discussion of their use is kept to a minimum. The next few chapters cover other portions of MS-DOS and compatible operating systems.

### 24-1 Overview of MS-DOS Interrupt Usage

| | |
|---|---|
| INT 20h | Terminate Program with PSP segment CS |
| INT 21h | DOS Functions |
| INT 22h | Termination Address |
| INT 23h | Ctrl-Break Interrupt |
| INT 24h | Critical Error Interrupt |
| INT 25h | Raw Disk Read |
| INT 26h | Raw Disk Write |
| INT 27h | Terminate and Stay Resident, keeping DX bytes |
| INT 28h | DOS Idle interrupt |
| INT 29h | Fast Console Output |
| INT 2Bh | Reserved, but not used |
| INT 2Ch | Reserved, but not used |
| INT 2Dh | Reserved, but not used |
| INT 2Fh | Multiplex |
| INT 30h | CP/M entry point JMP instruction |

### 24-2 Extended MS-DOS error codes:

| | | |
|---|---|---|
| 00h | (0) | no error |
| 01h | (1) | function number invalid |
| 02h | (2) | file not found |
| 03h | (3) | path not found |
| 04h | (4) | too many open files (no handles available) |
| 05h | (5) | access denied |
| 06h | (6) | invalid handle |
| 07h | (7) | memory control block destroyed |
| 08h | (8) | insufficient memory |
| 09h | (9) | memory block address invalid |
| 0Ah | (10) | environment invalid (usually >32K in length) |
| 0Bh | (11) | format invalid |
| 0Ch | (12) | access code invalid |

| | | |
|---|---|---|
| 0Dh | (13) | data invalid |
| 0Eh | (14) | reserved |
| 0Fh | (15) | invalid drive |
| 10h | (16) | attempted to remove current directory |
| 11h | (17) | not same device |
| 12h | (18) | no more files |
| **---DOS 3+---** | | |
| 13h | (19) | disk write-protected |
| 14h | (20) | unknown unit |
| 15h | (21) | drive not ready |
| 16h | (22) | unknown command |
| 17h | (23) | data error (CRC) |
| 18h | (24) | bad request structure length |
| 19h | (25) | seek error |
| 1Ah | (26) | unknown media type (non-DOS disk) |
| 1Bh | (27) | sector not found |
| 1Ch | (28) | printer out of paper |
| 1Dh | (29) | write fault |
| 1Eh | (30) | read fault |
| 1Fh | (31) | general failure |
| 20h | (32) | sharing violation |
| 21h | (33) | lock violation |
| 22h | (34) | disk change invalid |
| 23h | (35) | FCB unavailable |
| 24h | (36) | sharing buffer overflow |
| 25h | (37) | (DOS 4+) code page mismatch |
| 26h | (38) | (DOS 4+) cannot complete file operation (out of input) |
| 27h | (39) | (DOS 4+) insufficient disk space |
| 28h-31h | | reserved |
| 32h | (50) | network request not supported |
| 33h | (51) | remote computer not listening |
| 34h | (52) | duplicate name on network |
| 35h | (53) | network name not found |
| 36h | (54) | network busy |
| 37h | (55) | network device no longer exists |
| 38h | (56) | network BIOS command limit exceeded |
| 39h | (57) | network adapter hardware error |
| 3Ah | (58) | incorrect response from network |
| 3Bh | (59) | unexpected network error |
| 3Ch | (60) | incompatible remote adapter |
| 3Dh | (61) | print queue full |
| 3Eh | (62) | queue not full |
| 3Fh | (63) | not enough space to print file |
| 40h | (64) | network name was deleted |
| 41h | (65) | network: Access denied |
| 42h | (66) | network device type incorrect |
| 43h | (67) | network name not found |
| 44h | (68) | network name limit exceeded |
| 45h | (69) | network BIOS session limit exceeded |

| | | |
|---|---|---|
| 46h | (70) | temporarily paused |
| 47h | (71) | network request not accepted |
| 48h | (72) | network print/disk redirection paused |
| 49h | (73) | network software not installed<br>(LANtastic) invalid network version |
| 4Ah | (74) | unexpected adapter close<br>(LANtastic) account expired |
| 4Bh | (75) | (LANtastic) password expired |
| 4Ch | (76) | (LANtastic) login attempt invalid at this time |
| 4Dh | (77) | (LANtastic v3+) disk limit exceeded on network node |
| 4Eh | (78) | (LANtastic v3+) not logged in to network node |
| 4Fh | (79) | reserved |
| 50h | (80) | file exists |
| 51h | (81) | reserved |
| 52h | (82) | cannot make directory |
| 53h | (83) | fail on INT 24h |
| 54h | (84) | (DOS 3.3+) too many redirections |
| 55h | (85) | (DOS 3.3+) duplicate redirection passwords |
| 56h | (86) | (DOS 3.3+) invalid password |
| 57h | (87) | (DOS 3.3+) invalid parameter |
| 58h | (88) | (DOS 3.3+) network write fault |
| 59h | (89) | (DOS 4+) function not supported on network |
| 5Ah | (90) | (DOS 4+) required system component not installed<br>MSCDEX;error codes<br>CD-ROM;error codes<br>error codes;MSCDEX |
| 64h | (100) | (MSCDEX) unknown error |
| 65h | (101) | (MSCDEX) not ready |
| 66h | (102) | (MSCDEX) EMS memory no longer valid |
| 67h | (103) | (MSCDEX) not High Sierra or ISO-9660 format |
| 68h | (104) | (MSCDEX) door open |

## INT 21h Services

All of these calls potentially conflict with the Comtrol Hostess i/ISA Debugger (Chapter 57).

---

### INT 21h Function 00h

*DOS 1+*

***TERMINATE PROGRAM***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 00h<br>CS = PSP segment | never returns |

**Details:** Microsoft recommends using INT 21h Function 4Ch for DOS 2+. Execution continues at the address stored in INT 22h after DOS performs whatever cleanup it needs to do. If the PSP is its own parent, the process's memory is not freed; if INT 22h additionally points into the terminating program, the process is effectively *not t*erminated. This function is not supported by the MS Windows 3.0 DOSX.EXE DOS extender.

**See also:** Functions 26h, 31h, and 4Ch; INT 20h; INT 22h

---

### INT 21h Function 01h
DOS 1+

### *READ CHARACTER FROM STANDARD INPUT, WITH ECHO*

**Registers at call:**
AH = 01h

**Return Registers:**
AL = character read

**Details:** DOS checks for Control-C and Control-Break, and invokes INT 23h if either has been pressed.

The character is echoed to standard output.

Standard input is always the keyboard and standard output the screen under DOS 1.x, but they may be redirected under DOS 2+.

**See also:** Functions 06h, 07h, 08h, and 0Ah

---

### INT 21h Function 02h
DOS 1+

### *WRITE CHARACTER TO STANDARD OUTPUT*

**Registers at call:**
AH = 02h
DL = character to write

**Return Registers:**
AL = last character output (at least DOS 3.3-5.0)

**Details:** DOS checks for Control-C and Control-Break, and invokes INT 23h if either has been pressed.

A value is returned in AL despite official documents which state that nothin g is returned; the last character output will be the character in DL unless DL=09h on entry, in which case AL=20h as tabs are expanded to blanks.

**See also:** Function 06h and 09h

---

### INT 21h Functions 03h and 04h
DOS 1+

### *STDAUX CHARACTER I/O*

**Registers at call:**
AH = function
    03h read
    04h write
        DL = character to write

**Return Registers:**
AL = character read

**Details:** DOS checks for Control-C and Control-Break, and invokes INT 23h if either has been pressed.

STDAUX is usually the first serial port (COM1), but may be redirected.

If STDAUX is busy, Function 04h will wait until it becomes free.

**See also:** INT 14h Functions 01h and 02h (Chapter 12), DR Multiuser DOS INT E0h Functions 03h and 04h (Chapter 48)

---

### INT 21h Function 05h
DOS 1+

### *WRITE CHARACTER TO PRINTER*

**Registers at call:**
AH = 05h
DL = character to print

**Return Registers:**
none

**Details:** DOS checks for Control-C and Control-Break, and invokes INT 23h if either has been pressed. STDPRN is usually the first parallel port, but may be redirected under DOS 2+. If the printer is busy, this function will wait.

**See also:** INT 17h Function 00h (Chapter 15)

---

### INT 21h Function 06h
DOS 1+

### *DIRECT CONSOLE I/O*

**Registers at call:**
AH = 06h
DL = character to output, except FFh
DL = FFh input character

**Return Registers:**
AL = character output (at least DOS 3.3-5.0)
ZF set if no character available
ZF clear if character available
    AL = character read

**Details:** This function does not check for ^C/^Break.

When outputting a character, the character is sent to standard output, which is always the screen under DOS 1.x, but may be redirected under DOS 2+.

When inputting a character (DL=FFh), the character is read from standard input, which is always the keyboard under DOS 1.x, but may be redirected under DOS 2+. If the returned character is 00h, the user pressed a key with an extended keycode, which will be returned by the next call of this function.

The return value in AL on output is despite official documents which state that nothing is returned.

**See also:** Function 02h, 09h, and 0Bh

---

### INT 21h Function 07h *DOS 1+*
### *DIRECT CHARACTER INPUT, WITHOUT ECHO*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 07h | AL = character read from standard input |

**Details:** This function does not check for ^C/^Break.

If the interim console flag is set (see Function 6301h), partially-formed double-byte characters may be returned.

**See also:** Functions 01h, 06h, 08h, and 0Ah

---

### INT 21h Function 08h *DOS 1+*
### *CHARACTER INPUT WITHOUT ECHO*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 08h | AL = character read from standard input |

**Details:** DOS checks for Control-C and Control-Break, and invokes INT 23h if either has been pressed.

If the interim console flag is set (see Function 6301h), partially-formed double-byte characters may be returned.

**See also:** Functions 01h, 06h, 07h, 0Ah, and 64h

---

### INT 21h Function 09h *DOS 1+*
### *WRITE STRING TO STANDARD OUTPUT*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 09h | AL = 24h (the '$' terminating the string) |
| DS:DX -> '$'-terminated string | (at least DOS 3.3-5.0) |

**Details:** DOS checks for Control-C and Control-Break, and invokes INT 23h if either has been pressed.

The return value in AL is despite official documents which state that nothing is returned.

Under the FlashTek X-32 DOS extender, the pointer is in DS:EDX.

**See also:** Functions 02h and 06h

---

### INT 21h Function 0Ah *DOS 1+*
### *BUFFERED INPUT*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 0Ah | buffer filled with user input |
| DS:DX -> buffer (see Table 24-3) | |

**Details:** DOS checks for Control-C and Control-Break, and invokes INT 23h if either has been pressed.

This function reads from standard input, which may be redirected under DOS 2+.

If the maximum buffer size (see Table 24-3) is set to 00h, this call returns immediately without reading any input.

**Conflicts:** WCED (Chapter 13)

**See also:** Function 0Ch, INT 2Fh Function 4810h

**24-3 Format of input buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | maximum characters buffer can hold |
| 01h | BYTE | (input) number of characters from last input which may be recalled<br>(return) number of characters actually read, excluding CR |
| 02h | N BYTEs | actual characters read, including the final carriage return |

### INT 21h Function 0Bh
*DOS 1+*

***GET STDIN STATUS***

**Registers at call:**
AH = 0Bh

**Return Registers:**
AL = status
00h no character available
FFh character is available

**Details:** DOS checks for Control-C and Control-Break, and invokes INT 23h if either has been pressed.

When the interim console flag is set (see Function 6301h), this function returns AL=FFh if a partially-formed double-byte character is available.

**Conflicts:** Perfume virus (Chapter 59)

**See also:** Function 06h Subfunction FFh, Function 4406h

### INT 21h Function 0Ch
*DOS 1+*

***FLUSH BUFFER AND READ STANDARD INPUT***

**Registers at call:**
AH = 0Ch
AL = STDIN input function to execute after flushing buffer
other registers as appropriate for the input function

**Return Registers:**
as appropriate for the specified input function

**Details:** If AL is not one of 01h,06h,07h,08h, or 0Ah, the buffer is flushed but no input is attempted.

**See also:** Functions 01h, 06h, 07h, 08h, and 0Ah

### INT 21h Function 0Dh
*DOS 1+*

***DISK RESET***

**Registers at call:**
AH = 0Dh

**Return Registers:**
none

**Details:** DOS writes all modified disk buffers to disk, but does not update directory information (that is only done when files are closed or a SYNC call--Function 5D01h--is issued). Many disk caches monitor this function and force an immediate write of any modified but not yet written sectors.

**Conflicts:** Crazy Imp virus (Chapter 59)

**See also:** Function 5D01h, BIOS INT 13h Function 00h (Chapter 10), Network Redirector INT 2Fh Function 1120h (NI)

### INT 21h Function 0Eh
*DOS 1+*

***SELECT DEFAULT DRIVE***

**Registers at call:**
AH = 0Eh
DL = new default drive (00h = A:, 01h = B:, etc.)

**Return Registers:**
AL = number of potentially valid drive letters

**Details:** Under Novell NetWare, the return value is always 32, the number of drives that NetWare supports. Under DOS 3+, the return value is the greatest of 5, the value of LASTDRIVE= in CONFIG.SYS, and the number of drives actually present. On a single-floppy system, this function reports two drives since the floppy may be accessed as either A: or B:. Otherwise, the return value is the highest drive actually present.

DOS 1.x supports a maximum of 16 drives, 2.x a maximum of 63 drives, and 3+ a maximum of 26 drives.

**See also:** Functions 19h and 3Bh, NetWare Function DBh (NI)

---

## INT 21h Function 0Fh *DOS 1+*

### *OPEN FILE USING FCB*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 0Fh | AL = status |
| DS:DX -> unopened File Control Block (see Table 24-4) | 00h successful<br>FFh file not found or access denied |

**Details:** Under DOS 3.1+, the file is opened for read/write in compatibility mode.

An unopened FCB has the drive, filename, and extension fields filled in and all other bytes cleared.

DR-DOS checks the password attached with Function 4303h.

This function is not supported by the MS Windows 3.0 DOSX.EXE DOS extender.

**See also:** Functions 10h, 16h, and 3Dh; DR-DOS Function 4303h (Chapter 26)

**24-4 Format of File Control Block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -7 | BYTE | extended FCB if FFh |
| -6 | 5 BYTEs | reserved |
| -1 | BYTE | file attribute if extended FCB |
| 00h | BYTE | drive number (0 = default, 1 = A, etc.) |
| 01h | 8 BYTEs | blank-padded file name |
| 09h | 3 BYTEs | blank-padded file extension |
| 0Ch | WORD | current block number |
| 0Eh | WORD | logical record size |
| 10h | DWORD | file size |
| 14h | WORD | date of last write (see Table 24-97) |
| 16h | WORD | time of last write (see Table 24-96) (DOS 1.1+) |
| 18h | 8 BYTEs | reserved (see Table 24-5) |
| 20h | BYTE | record within current block |
| 21h | DWORD | random access record number (if record size is > 64 bytes, high byte is omitted) |

**Note:** To use an extended FCB, you must specify the address of the FFh flag at offset -7, rather than the address of the drive number field.

**24-5 Format of reserved field for DOS 3.x:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 18h | BYTE | number of system file table entry for file |
| 19h | BYTE | attributes<br>bits 7,6: 00 = SHARE.EXE not loaded, disk file<br>01 = SHARE.EXE not loaded, character device<br>10 = SHARE.EXE loaded, remote file<br>11 = SHARE.EXE loaded, local file or device<br>bits 5-0: low six bits of device attribute word |
| **---SHARE.EXE loaded, local file---** | | |
| 1Ah | WORD | starting cluster of file on disk |
| 1Ch | WORD | (DOS 3.x) offset within SHARE of sharing record (see Table 24-77) |
| 1Eh | BYTE | file attribute |
| 1Fh | BYTE | *unknown* |

| | | |
|---|---|---|
| **---SHARE.EXE loaded, remote file---** | | |
| 1Ah | WORD | number of sector containing directory entry |
| 1Ch | WORD | relative cluster within file of last cluster accessed |
| 1Eh | BYTE | absolute cluster number of last cluster accessed |
| 1Fh | BYTE | *unknown* |
| **---SHARE.EXE not loaded---** | | |
| 1Ah | BYTE | (low byte of device attribute word AND 0Ch) OR open mode |
| 1Bh | WORD | starting cluster of file |
| 1Dh | WORD | number of sector containing directory entry |
| 1Fh | BYTE | number of directory entry within sector |

**Notes:** If the FCB was opened on a character device, the DWORD at 1Ah is set to the address of the device driver header, and then the BYTE at 1Ah is overwritten.

The reserved fields for DOS 1.0, 1.1-1.25, and 2.x have been omitted due to lack of space; they may be found in *Network Interrupts* or *Undocumented DOS.*

**24-6 Format of reserved field for DOS 5.0:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 18h | BYTE | number of system file table entry for file |
| 19h | BYTE | attributes<br>bits 7,6: 00 = SHARE.EXE not loaded, disk file<br>01 = SHARE.EXE not loaded, character device<br>10 = SHARE.EXE loaded, remote file<br>11 = SHARE.EXE loaded, local file or device<br>bits 5-0: low six bits of device attribute word |
| **---SHARE.EXE loaded, local file---** | | |
| 1Ah | WORD | starting cluster of file on disk |
| 1Ch | WORD | unique sequence number of sharing record |
| 1Eh | BYTE | file attributes |
| 1Fh | BYTE | *unused* |
| **---SHARE.EXE loaded, remote file---** | | |
| 1Ah | WORD | network handle |
| 1Ch | DWORD | network ID |
| **---SHARE not loaded, local device---** | | |
| 1Ah | DWORD | pointer to device driver header |
| 1Eh 2 | BYTEs | *unused* |
| **---SHARE not loaded, local file---** | | |
| 1Ah | BYTE | extra info<br>bit 7: read-only attribute from SFT<br>bit 6: archive attribute from SFT<br>bits 5-0: high bits of sector number |
| 1Bh | WORD | starting cluster of file |
| 1Dh | WORD | low word of sector number containing directory entry |
| 1Fh | BYTE | number of directory entry within sector |

## INT 21h Function 10h
### *CLOSE FILE USING FCB*

*DOS 1+*

**Registers at call:**
AH = 10h
DS:DX -> File Control Block
(see Table 24-4)

**Return Registers:**
AL = status
(00h successful, FFh failed)

**Details:** A successful close forces all disk buffers used by the file to be written and the directory entry to be updated.

This function is not supported by the MS Windows 3.0 DOSX.EXE DOS extender.

**See also:** Functions 0Fh, 16h, and 3Eh

---

### INT 21h Function 11h

*DOS 1+*

### *FIND FIRST MATCHING FILE USING FCB*

**Registers at call:**

AH = 11h
DS:DX -> unopened FCB (see Table 24-4), may contain '?' wildcards

**Return Registers:**

AL = status
  00h successful
    [DTA] unopened FCB for first matching file
  FFh no matching filename, or bad FCB

**Details:** The type of the returned FCB depends on whether the input FCB was a normal or an extended FCB.

The data returned in the DTA is actually the drive number (or extended FCB header and drive number) followed by the file's directory entry (see Table 24-7); this format happens to be compatible with an unopened FCB.

For extended FCBs with search attribute 08h, the volume label (if any) will be returned even if the current directory is not the root dir.

DOS 3+ also allows the '*' wildcard in addition to question marks.

The search FCB must not be modified if Function 12h will be used to continue searching; DOS 3.3 has set the following parts of the FCB:

| | | |
|---|---|---|
| 0Ch | BYTE | *unknown* |
| 0Dh | WORD | directory entry number of matching file |
| 0Fh | WORD | cluster number of current directory |
| 11h | 4 BYTEs | *unknown* |
| 15h | BYTE | drive number (1=A:). |

This function is used by many copy protection schemes to obtain the starting cluster of a file.

**See also:** Functions 12h, 1Ah, and 4Eh; Network Redirector INT 2Fh Function 111Bh (NI)

**24-7 Format of directory entry:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 8 BYTEs | blank-padded filename |
| 08h | 3 BYTEs | blank-padded file extension |
| 0Bh | BYTE | attributes |
| 0Ch | 10 BYTEs | reserved (used by DR-DOS to store the file password) |
| 16h | WORD | time of creation or last update (see Table 24-96) |
| 18h | WORD | date of creation or last update (see Table 24-97) |
| 1Ah | WORD | starting cluster number |
| 1Ch | DWORD | file size |

---

### INT 21h Function 12h

*DOS 1+*

### *FIND NEXT MATCHING FILE USING FCB*

**Registers at call:**

AH = 12h
DS:DX -> unopened FCB (see Table 24-4)

**Return Registers:**

AL = status
  00h successful
    [DTA] = unopened FCB
  FFh no more matching filenames

**Details:** (see Function 11h).

This function assumes that a successful FindFirst was executed on the search FCB before being called; results are undefined if this is not the case or the previous FindNext call indicated no more matches.

**See also:** Functions 1Ah and 4Fh, Network Redirector INT 2Fh Function 111Ch (NI)

## INT 21h Function 13h

*DOS 1+*

### *DELETE FILE USING FCB*

**Registers at call:**

AH = 13h

DS:DX -> unopened FCB (see Table 24-4), filename filled with template for deletion ('?' wildcards allowed)

**Return Registers:**

AL = status

- 00h one or more files successfully deleted
- FFh no matching files or all were read-only or locked

**Details:** DOS 1.25+ deletes everything in the current directory (including subdirectories) and sets the first byte of the name to 00h (entry never used) instead of E5h if called on an extended FCB with filename '???????????' and bits 0-4 of the attribute set (bits 1 and 2 for DOS 1.x). This may have originally been an optimization to minimize directory searching after a mass deletion (DOS 1.25+ stop the directory search upon encountering a never-used entry), but can corrupt the filesystem under DOS 2+ because subdirectories are removed without deleting the files they contain.

Currently-open files should not be deleted.

MS-DOS allows deletion of read-only files with an extended FCB, whereas Novell NetWare does not.

**See also:** Function 41h, INT 2Fh Function 1113h

## INT 21h Function 14h

*DOS 1+*

### *SEQUENTIAL READ FROM FCB FILE*

**Registers at call:**

AH = 14h

DS:DX -> opened FCB (see Table 24-4)

**Return Registers:**

AL = status

- 00h successful
- 01h end of file (no data)
- 02h segment wrap in DTA
- 03h end of file, partial record read

[DTA] = record read from file

**Details:** This function reads a record of the size specified in the FCB beginning at the current file position, then updates the current block and current record fields in the FCB.

If a partial record was read, it is zero-padded to the full size.

This function is not supported by the MS Windows 3.0 DOSX.EXE DOS extender.

**See also:** Functions 0Fh, 15h, 1Ah, and 3Fh; Network Redirector INT 2Fh Function 1108h (NI)

## INT 21h Function 15h

*DOS 1+*

### *SEQUENTIAL WRITE TO FCB FILE*

**Registers at call:**

AH = 15h

DS:DX -> opened FCB (see Table 24-4)

[DTA] = record to write

**Return Registers:**

AL = status

- 00h successful
- 01h disk full
- 02h segment wrap in DTA

**Details:** This function writes a record of the size specified in the FCB beginning at the current file position, then updates the current block and current record fields in the FCB.

If less than a full sector is written, the data is placed in a DOS buffer to be written out at a later time.

This function is not supported by the MS Windows 3.0 DOSX.EXE DOS extender.

**See also:** Functions 0Fh, 14h, 1Ah, and 40h; Network Redirector INT 2Fh Function 1109h (NI)

---

**INT 21h Function 16h** *DOS 1+*

***CREATE OR TRUNCATE FILE USING FCB***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 16h | AL = status |
| DS:DX -> unopened FCB (see Table 24-4), wildcards not allowed | 00h successful<br>FFh directory full or file exists and is read-only or locked |

**Details:** If the file already exists, it is truncated to zero length.

If an extended FCB is used, the file is given the attribute in the FCB; this is how to create a volume label in the disk's root directory.

This function is not supported by the MS Windows 3.0 DOSX.EXE DOS extender.

**See also:** Functions 0Fh, 10h, and 3Ch

---

**INT 21h Function 17h** *DOS 1+*

***RENAME FILE USING FCB***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 17h | AL = status |
| DS:DX -> modified FCB (see also Table 24-4) | 00h successfully renamed<br>FFh no matching files, file is read-only, or new name already exists |

**Details:** The old filename is in the standard location while the new filename is stored in the 11 bytes beginning at offset 11h. Both the old and new filenames may contain '?' wildcards, where wildcards in the new filename are copied from each matched filename during renaming.

Subdirectories may be renamed using an extended FCB with the appropriate attribute, as may volume labels.

DR-DOS checks the password attached with Function 4303h before permitting the rename.

**See also:** Functions 0Fh, 13h, and 56h; DR-DOS Function 4303h (Chapter 26); Network Redirector INT 2Fh Function 1111h (NI)

---

**INT 21h Function 18h** *DOS 1+*

***NULL FUNCTION FOR CP/M COMPATIBILITY***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 18h | AL = 00h |

**Details:** This function corresponds to the CP/M BDOS function "get bit map of logged drives", which is meaningless under MS-DOS.

**See also:** Functions 1Dh, 1Eh, and 20h; DR-DOS Function 4459h (Chapter 26)

---

**INT 21h Function 19h** *DOS 1+*

***GET CURRENT DEFAULT DRIVE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 19h | AL = drive (00h = A:, 01h = B:, etc.) |

**Details:** Novell NetWare uses the fact that DOS 2.x COMMAND.COM issues this call from a particular location every time it starts a command to determine when to issue an automatic EOJ (End of Job).

**See also:** Functions 0Eh and 47h, Novell NetWare Function BBh (NI)

---

**INT 21h Function 1Ah** *DOS 1+*

***SET DISK TRANSFER AREA ADDRESS***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 1Ah | none |
| DS:DX -> Disk Transfer Area (DTA) | |

**Details:** The DTA is set to PSP:0080h when a program is started.

Under the FlashTek X-32 DOS extender, the pointer is in DS:EDX.

**See also:** Functions 11h, 12h, 2Fh, 4Eh, and 4Fh

## INT 21h Functions 1Bh and 1Ch

DOS 1+

### *GET ALLOCATION INFORMATION FOR DEFAULT/SPECIFIED DRIVE*

**Registers at call:**
AH = function
1Bh default drive
1Ch specified drive
DL = drive (00h default, 01h A:, etc.)

**Return Registers:**
AL = sectors per cluster (allocation unit), or FFh if invalid drive
CX = bytes per sector
DX = total number of clusters
DS:BX -> media ID byte (see Table 24-8)

**Details:** Under DOS 1.x, DS:BX points at an actual copy of the FAT; later versions return a pointer to a copy of the FAT's ID byte.

On a DBLSPACE drive, the total number of clusters is based on the estimated compression ratio.

**See also:** Function 36h

**24-8 Values for media ID byte:**

| | |
|---|---|
| FFh | floppy, double-sided, 8 sectors per track (320K) |
| FEh | floppy, single-sided, 8 sectors per track (160K) |
| FDh | floppy, double-sided, 9 sectors per track (360K) |
| FCh | floppy, single-sided, 9 sectors per track (180K) |
| F9h | floppy, double-sided, 15 sectors per track (1.2M) |
| F8h | hard disk |
| F0h | other |

## INT 21h Functions 1Dh and 1Eh

DOS 1+

### *NULL FUNCTIONS FOR CP/M COMPATIBILITY*

**Registers at call:**
AH = 1Dh and 1Eh

**Return Registers:**
AL = 00h

**Details:** These functions correspond to the CP/M BDOS functions "get bit map of read-only drives" and "set file attributes", which are meaningless under MS-DOS 1.x.

**See also:** Functions 18h, 1Eh, and 20h; DR-DOS Function 4459h (Chapter 26)

## INT 21h Function 1Fh

DOS 1+

### *GET DRIVE PARAMETER BLOCK FOR DEFAULT DRIVE*

**Registers at call:**
AH = 1Fh

**Return Registers:**
AL = status
00h successful
DS:BX -> Drive Parameter Block (DPB) (see Table 24-15 for DOS 2+)
FFh invalid drive

**Details:** This call was undocumented prior to the release of DOS 5.0; however, only the DOS 4+ version of the DPB has been documented.

The DOS 1.x DPB formats have been omitted due to lack of space; they may be found in *Undocumented DOS*.

**See also:** Function 32h

## INT 21h Function 20h

DOS 1+

### *NULL FUNCTION FOR CP/M COMPATIBILITY*

**Registers at call:**
AH = 20h

**Return Registers:**
AL = 00h

**Details:** This function corresponds to the CP/M BDOS function "get/set default user (sublibrary) number", which is meaningless under MS-DOS.

**See also:** Functions 18h, 1Dh, and 1Eh; DR-DOS Function 4459h (Chapter 26)

### INT 21h Function 21h

DOS 1+

### *READ RANDOM RECORD FROM FCB FILE*

**Registers at call:**
AH = 21h
DS:DX -> opened FCB (see Table 24-4)

**Return Registers:**
AL = status (see Table 24-9)
[DTA] = record read from file

**Details:** The record is read from the current file position as specified by the random record and record size fields of the FCB; the file position is not updated after reading the record. If a partial record is read, it is zero-padded to the full size.

This function is not supported by the MS Windows 3.0 DOSX.EXE DOS extender.

**See also:** Functions 14h, 22h, 27h, and 3Fh

**24-9 Values for status:**

| | |
|---|---|
| 00h | successful |
| 01h | end of file, no data read |
| 02h | segment wrap in DTA, no data read |
| 03h | end of file, partial record read |

### INT 21h Function 22h

DOS 1+

### *WRITE RANDOM RECORD TO FCB FILE*

**Registers at call:**
AH = 22h
DS:DX -> opened FCB (see Table 24-4)
[DTA] = record to write

**Return Registers:**
AL = status (see Table 24-10)

**Details:** The record is written to the current file position as specified by the random record and record size fields of the FCB; the file position is not updated after writing the record.

If the record is located beyond the end of the file, the file is extended but the intervening data remains uninitialized. If the record only partially fills a disk sector, it is copied to a DOS disk buffer to be written out to disk at a later time.

This function is not supported by the MS Windows 3.0 DOSX.EXE DOS extender.

**See also:** Functions 15h, 21h, 28h, and 40h

**24-10 Values for status:**

| | |
|---|---|
| 00h | successful |
| 01h | disk full |
| 02h | segment wrap in DTA |

### INT 21h Function 23h

DOS 1+

### *GET FILE SIZE FOR FCB*

**Registers at call:**
AH = 23h
DS:DX -> unopened FCB (see Table 24-4), wildcards not allowed

**Return Registers:**
AL = status
  00h successful (matching file found)
    FCB random record field filled with size in records, rounded up to next full record
  FFh failed (no matching file found)

**Details:** This function is not supported by the MS Windows 3.0 DOSX.EXE DOS extender.

MS-DOS returns nonsense if the FCB record number field is set to a very large positive number, and status FFh if negative; DR-DOS returns the correct file size in both cases.

**See also:** Function 42h

### INT 21h Function 24h

DOS 1+

### *SET RANDOM RECORD NUMBER FOR FCB*

**Registers at call:**
AH = 24h
DS:DX -> opened FCB (see Table 24-4)

**Return Registers:**
none

**Details:** This function computes the random record number corresponding to the current record number and record size, then stores the result in the FCB. It is normally used when switching from sequential to random access.

This function is not supported by the MS Windows 3.0 DOSX.EXE DOS extender.

**See also:** Functions 21h, 27h, and 42h

---

### INT 21h Function 25h
### *SET INTERRUPT VECTOR*

*DOS 1+*

**Registers at call:**
AH = 25h
AL = interrupt number
DS:DX -> new interrupt handler

**Return Registers:**
none

**Details:** This function is preferred over direct modification of the interrupt vector table. Under DR-DOS 5.0+, this function does not use any of the DOS-internal stacks and may thus be called at any time.

Some DOS extenders such as Phar Lap's (Chapter 33) place an API on this function, since it is not directly meaningful in protected mode.

Novell NetWare (except the new DOS Requester) monitors the offset of any INT 24h set, and if equal to the value at startup, substitutes its own handler to allow handling of network errors; this introduces the potential bug that any program whose INT 24h handler offset happens to be the same as COMMAND.COM's will not have its INT 24h handler installed.

**Conflicts:** Phar Lap 386|DOS-Extender and FlashTek X-32VM (Chapter 33)

**See also:** Function 35h, Phar Lap Function 2501h (Chapter 33)

---

### INT 21h Function 26h
### *CREATE NEW PROGRAM SEGMENT PREFIX*

*DOS 1+*

**Registers at call:**
AH = 26h
DX = segment at which to create PSP
(see Table 24-11)

**Return Registers:**
none

**Details:** The new PSP is updated with memory size information; INTs 22h, 23h, 24h are taken from interrupt vector table.

DOS 2+ assumes that the caller's CS is the segment of the PSP to copy.

**See also:** Functions 4Bh, 50h, 51h, 55h, 62h, and 67h

**24-11 Format of PSP:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | INT 20h instruction for CP/M CALL 0 program termination.<br>The CDh 20h here is often used as a signature for a valid PSP. |
| 02h | WORD | segment of first byte beyond memory allocated to program |
| 04h | BYTE | unused filler |
| 05h | BYTE | CP/M CALL 5 service request (FAR JMP to 000C0h, see INT 30h below)<br>**BUG:** (DOS 2-6) PSPs created by INT 21h Function 4Bh point at 000BEh |
| 06h | WORD | CP/M compatibility--size of first segment for .COM files |
| 08h | 2 BYTEs | remainder of FAR JMP at 05h |
| 0Ah | DWORD | stored INT 22h termination address |
| 0Eh | DWORD | stored INT 23h control-Break handler address |
| 12h | DWORD | DOS 1.1+ stored INT 24h critical error handler address |
| 16h | WORD | segment of parent PSP |
| 18h | 20 BYTEs | DOS 2+ Job File Table, one byte per file handle, FFh = closed |
| 2Ch | WORD | DOS 2+ segment of environment for process |
| 2Eh | DWORD | DOS 2+ process's SS:SP on entry to last INT 21h call |

| | | |
|---|---|---|
| 32h | WORD | DOS 3+ number of entries in JFT (default 20) |
| 34h | DWORD | DOS 3+ pointer to JFT (default PSP:0018h) |
| 38h | DWORD | DOS 3+ pointer to previous PSP (default FFFFFFFFh in 3.x)<br>used by SHARE in DOS 3.3 |
| 3Ch | BYTE | apparently unused by DOS versions <= 6.00 |
| 3Dh | BYTE | apparently used by some versions of APPEND |
| 3Eh | BYTE | (Novell NetWare) flag: next byte initialized if CEh |
| 3Fh | BYTE | (Novell NetWare) Novell task number if previous byte is CEh |
| 40h | 2 BYTEs | DOS 5+ version to return on INT 21h Function 30h (SETVER) |
| 42h | WORD | (MSWin3) selector of next PSP (PDB) in linked list<br>Windows keeps a linked list of Windows programs only |
| 44h | 4 BYTEs | unused by DOS versions <= 6.00 |
| 48h | BYTE | (MSWindows3) bit 0 set if non-Windows application (WINOLDAP) |
| 49h | 7 BYTEs | unused by DOS versions <= 6.00 |
| 50h | 3 BYTEs | DOS 2+ service request (INT 21h/RETF instructions) |
| 53h | 2 BYTEs | unused in DOS versions <= 6.00 |
| 55h | 7 BYTEs | unused in DOS versions <= 6.00; can be used to make first FCB into an extended FCB |
| 5Ch | 16 BYTEs | first default FCB, filled in from first commandline argument<br>overwrites second FCB if opened |
| 6Ch | 16 BYTEs | second default FCB, filled in from second commandline argument<br>overwrites beginning of commandline if opened |
| 7Ch | 4 BYTEs | unused |
| 80h | 128 BYTEs | commandline and default DTA<br>the command tail is a BYTE for the length of the tail, N BYTEs for the tail, followed by a BYTE containing 0Dh |

**Notes:** In DOS v3+, the limit on simultaneously open files may be increased by allocating memory for a new open file table, filling it with FFh, copying the first 20 bytes from the default table, and adjusting the pointer and count at 34h and 32h. However, DOS will only copy the first 20 file handles into a child PSP (including the one created on EXEC).

Network redirectors based on the original MS-Net implementation use values of 80h-FEh in the open file table to indicate remote files; Novell NetWare reportedly also uses values of 80h-FEh.

MS-DOS 5.00 incorrectly fills the FCB fields when loading a program high; the first FCB is empty and the second contains the first parameter.

Some DOS extenders place protected-mode values in various PSP fields such as the "parent" field, which can confuse PSP walkers. Always check either for the CDh 20h signature or that the suspected PSP is at the beginning of a memory block which owns itself (the preceding paragraph should be a valid MCB with "owner" the same as the suspected PSP).

Novell NetWare updates the fields at offsets 3Eh and 3Fh without checking that a legal PSP segment is current; see Function 50h for further discussion.

**24-12 Format of environment block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | N BYTEs | first environment variable, ASCIZ string of form "var=value" |
| | N BYTEs | second environment variable, ASCIZ string |
| | ... | |
| | N BYTEs | last environment variable, ASCIZ string of form "var=value" |
| | BYTE | 00h |
| **---DOS 3+---** | | |
| | WORD | number of strings following environment (normally 1) |
| | N BYTEs | ASCIZ full pathname of program owning this environment; other strings may follow |

---

## INT 21h Function 27h

*DOS 1+*

### *RANDOM BLOCK READ FROM FCB FILE*

**Registers at call:**
AH = 27h
CX = number of records to read
DS:DX -> opened FCB (see Table 24-4)

**Return Registers:**
AL = status (see Table 24-9)
[DTA] = records read from file
CX = number of records read (return AL = 00h or 03h)

**Details:** Reading begins at the current file position as specified in the FCB; the file position is updated after reading.

This function is not supported by the MS Windows 3.0 DOSX.EXE DOS extender.

**See also:** Functions 21h, 28h, and 3Fh

---

## INT 21h Function 28h

*DOS 1+*

### *RANDOM BLOCK WRITE TO FCB FILE*

**Registers at call:**
AH = 28h
CX = number of records to write
DS:DX -> opened FCB (see Table 24-4)
[DTA] = records to write

**Return Registers:**
AL = status (see Table 24-10)
CX = number of records written

**Details:** Writing begins at the current file position as specified in the FCB; the file position is updated after writing. If the data to be written is less than a disk sector, it is copied into a DOS disk buffer, to be written out to disk at a later time.

If CX = 0000h on entry, no data is written; instead the file size is adjusted to be the same as the file position specified by the random record and record size fields of the FCB.

This function is not supported by the MS Windows 3.0 DOSX.EXE DOS extender.

**See also:** Functions 22h, 27h, 40h, and 59h

---

## INT 21h Function 29h

*DOS 1+*

### *PARSE FILENAME INTO FCB*

**Registers at call:**
AH = 29h
AL = parsing options (see Table 24-13)
DS:SI -> filename string (both ’*’ and ’?’ wildcards OK)
ES:DI -> buffer for unopened FCB

**Return Registers:**
AL = result code
- 00h successful parse, no wildcards encountered
- 01h successful parse, wildcards present
- FFh failed (invalid drive specifier)

DS:SI -> first unparsed character
ES:DI buffer filled with unopened FCB (see Table 24-4)

**Details:** Asterisks are expanded to question marks in the FCB.

All processing stops when a filename terminator is encountered.

This function can not be used with filespecs which include a path (DOS 2+).

Novell NetWare monitors the result code since an ’invalid drive’ may signal an attempt to reconnect a network drive; if there are no connections to the specified drive, NetWare attempts to build a connection and map the drive to the SYS:LOGIN directory.

**See also:** Functions 0Fh, 16h, and 26h

**24-13 Bitfields for parsing options:**

| | |
|---|---|
| bit 0 | skip leading separators |
| bit 1 | use existing drive number in FCB if no drive is specified, instead of setting field to zero |
| bit 2 | use existing filename in FCB if no base name is specified, instead of filling field with blanks |
| bit 3 | use existing extension in FCB if no extension is specified, instead of filling field with blanks |
| bits 4-7 | reserved (0) |

---

### INT 21h Function 2Ah
*DOS 1+*

### *GET SYSTEM DATE*

**Registers at call:**
AH = 2Ah

**Return Registers:**
CX = year (1980-2099)
DH = month
DL = day
AL = day of week (00h=Sunday) (DOS 1.10+)

**See also:** Functions 2Bh and 2Ch, INT 2Fh Function 120Dh (below), BIOS INT 1Ah Function 04h (Chapter 4), Novell NetWare Function E7h (NI)

---

### INT 21h Function 2Bh
*DOS 1+*

### *SET SYSTEM DATE*

**Registers at call:**
AH = 2Bh
CX = year (1980-2099)
DH = month
DL = day

**Return Registers:**
AL = status
 00h successful
 FFh invalid date, system date unchanged

**Details:** DOS 3.3+ also sets the CMOS clock.

**Conflicts:** Super PC-Kwik (Chapter 17), AI Architects (Chapter 37), DESQview (Chapter 44), pcANYWHERE (Chapter 58), DOS-UP.SYS (Chapter 63), ELRES (Chapter 66), TAME (Chapter 66)

**See also:** Functions 2Ah and 2Dh, BIOS INT 1Ah Function 05h (Chapter 4)

---

### INT 21h Function 2Ch
*DOS 1+*

### *GET SYSTEM TIME*

**Registers at call:**
AH = 2Ch

**Return Registers:**
CH = hour
CL = minute
DH = second
DL = 1/100 seconds

**Details:** On most systems, the resolution of the system clock is about 5/100sec, so returned times generally do not increment by 1.

On some systems, DL may always return 00h.

**See also:** Functions 2Ah and 2Dh, INT 2Fh Function 120Dh, BIOS INT 1Ah Functions 00h and 02h (Chapter 4), AT&T INT 1Ah Function FEh (Chapter 7), Novell NetWare Function E7h (NI)

---

### INT 21h Function 2Dh
*DOS 1+*

### *SET SYSTEM TIME*

**Registers at call:**
AH = 2Dh
CH = hour
CL = minute
DH = second
DL = 1/100 seconds

**Return Registers:**
AL = result
 00h successful
 FFh invalid time, system time unchanged

**Details:** DOS 3.3+ also sets the CMOS clock.

**Conflicts:** PC-Mix (Chapter 49)

**See also:** Functions 2Bh and 2Ch, BIOS INT 1Ah Functions 01h and 03h (Chapter 4), AT&T INT 1Ah Function FFh (Chapter 7)

## INT 21h Function 2Eh

*DOS 1+*

### *SET VERIFY FLAG*

**Registers at call:**
AH = 2Eh
DL = 00h (DOS 1.x/2.x only)
AL = new state of verify flag
(00h off, 01h on)

**Return Registers:**
none

**Details:** The default state at system boot is *off.* When *on*, all disk writes are verified provided the device driver supports read-after-write verification.

**See also:** Function 54h

## INT 21h Function 2Fh

*DOS 2+*

### *GET DISK TRANSFER AREA ADDRESS*

**Registers at call:**
AH = 2Fh

**Return Registers:**
ES:BX -> current DTA

**Details:** Under the FlashTek X-32 DOS extender, the pointer is returned in ES:EBX.

**See also:** Function 1Ah

## INT 21h Function 30h

*DOS 2+*

### *GET DOS VERSION*

**Registers at call:**
AH = 30h
---**DOS 5+** ---
AL = what to return in BH
00h OEM number (as for DOS 2.0-4.0x)
01h version flag

**Return Registers:**
AL = major version number (00h if DOS 1.x)
AH = minor version number
BL:CX = 24-bit user serial number (most versions do not use this)
---**if DOS <5 or AL=00h**---
BH = MS-DOS OEM number (see Table 24-14)
---**if DOS 5+ and AL=01h**---
BH = version flag
bit 3: DOS is in ROM
other: reserved (0)

**Details:** The OS/2 v1.x Compatibility Box returns major version 0Ah (10), the OS/2 v2.x Compatibility Box returns major version 14h (20), and the Windows/NT DOS box returns version 5.00, subject to SETVER.

Generic MS-DOS 3.30, Compaq MS-DOS 3.31, and others identify themselves as PC-DOS by returning OEM number 00h.

DOS 4.01 and 4.02 identify themselves as version 4.00; use INT 21h Function 87h to distinguish between the original European MS-DOS 4.0 and the later PC-DOS 4.0x and MS-DOS 4.0x.

The version returned under DOS 4.0x may be modified by entries in the special program list (see Table 24-94); the version returned under DOS 5+ may be modified by SETVER--use Function 3306h to get the true version number.

IBM DOS 6.10 reports its version as 6.00.

**Conflicts:** Phar Lap 386|DOS-Extender and Intel Code Builder (Chapter 33), DESQ (Chapter 44), CTask (Chapter 49), PC-MOS/386 (Chapter 49), StopPrg (Chapter 57), viruses (Chapter 59)

**See also:** Function 3306h, INT 2Fh Function 122Fh (below), Far East MS-DOS INT 15h Function 4900h (below), PC-MOS/386 Function 3000h Subfunction 3000h (Chapter 49), DR-DOS Function 4452h (Chapter 26), European MS-DOS Function 87h (Chapter 25), SETDRVER INT 2Fh Function E002h (Chapter 66)

**24-14 MS-DOS OEM numbers:**

| | |
|---|---|
| 00h | IBM |
| 01h | Compaq |
| 02h | MS Packaged Product |
| 04h | AT&T |
| 05h | Zenith |
| 06h | Hewlett-Packard |
| 0Dh | Packard-Bell |
| 16h | DEC |
| 23h | Olivetti |
| 29h | Toshiba |
| 33h | Novell (Windows/386 device IDs only) |
| 34h | MS Multimedia Systems (Windows/386 device IDs only) |
| 35h | MS Multimedia Systems (Windows/386 device IDs only) |
| 4Dh | Hewlett-Packard |
| 66h | PhysTechSoft (PTS-DOS) |
| 99h | General Software's Embedded DOS |
| FFh | Microsoft, Phoenix |

---

## INT 21h Function 31h

*DOS 2+*

### *TERMINATE AND STAY RESIDENT*

**Registers at call:**
AH = 31h
AL = return code
DX = number of paragraphs to keep resident

**Return Registers:**
never returns

**Details:** The value in DX only affects the memory block containing the PSP; additional memory allocated via Function 48h is not affected.

The minimum number of paragraphs which will remain resident is 11h for DOS 2.x and 06h for DOS 3+.

Most TSRs can save some memory by releasing their environment block before terminating (see Functions 26h and 49h).

**See also:** Functions 00h, 4Ch, and 4Dh; INT 20h; INT 22h; INT 27h

---

## INT 21h Function 32h

*DOS 2+*

### *GET DOS DRIVE PARAMETER BLOCK FOR SPECIFIC DRIVE*

**Registers at call:**
AH = 32h
DL = drive number (00h default, 01h A:, etc.)

**Return Registers:**
AL = status
  00h successful
    DS:BX -> Drive Parameter Block (DPB) for specified drive
  FFh invalid or network drive

**Details:** The OS/2 compatibility box supports the DOS 3.3 version of this call except for the DWORD at offset 12h. DR-DOS 3.41-6.0 return the same data as MS-DOS 3.31.

This call updates the DPB by reading the disk; the DPB may be accessed via the DOS list of lists (see Function 52h) if disk access is not desirable.

This function was undocumented prior to the release of DOS 5.0; only the DOS 4+ version of the DPB has been documented, however.

**See also:** Functions 1Fh and 52h

### 24-15 Format of DOS 2+ Drive Parameter Block:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | drive number (00h = A:, 01h = B:, etc.) |
| 01h | BYTE | unit number within device driver |
| 02h | WORD | bytes per sector |
| 04h | BYTE | highest sector number within a cluster |
| 05h | BYTE | shift count to convert clusters into sectors |
| 06h | WORD | number of reserved sectors at beginning of drive |
| 08h | BYTE | number of FATs |
| 09h | WORD | number of root directory entries |
| 0Bh | WORD | number of first sector containing user data |
| 0Dh | WORD | highest cluster number (number of data clusters + 1)<br>16-bit FAT if greater than 0FF6h, else 12-bit FAT |
| 0Fh | BYTE | number of sectors per FAT |
| 10h | WORD | sector number of first directory sector |
| 12h | DWORD | address of device driver header |
| 16h | BYTE | media ID byte |
| 17h | BYTE | 00h if disk accessed, FFh if not |
| 18h | DWORD | pointer to next DPB |
| ---DOS 2.x--- | | |
| 1Ch | WORD | cluster containing start of current directory, 0000h=root, FFFFh = unknown |
| 1Eh | 64 BYTEs | ASCIZ pathname of current directory for drive |
| ---DOS 3.x--- | | |
| 1Ch | WORD | cluster at which to start search for free space when writing |
| 1Eh | WORD | number of free clusters on drive, FFFFh = unknown |
| ---DOS 4.0-6.0--- | | |
| 0Fh | WORD | number of sectors per FAT |
| 11h | WORD | sector number of first directory sector |
| 13h | DWORD | address of device driver header |
| 17h | BYTE | media ID byte |
| 18h | BYTE | 00h if disk accessed, FFh if not |
| 19h | DWORD | pointer to next DPB |
| 1Dh | WORD | cluster at which to start search for free space when writing, usually the last cluster allocated |
| 1Fh | WORD | number of free clusters on drive, FFFFh = unknown |

## INT 21h Functions 3300h and 3301h

*DOS 2+*

### *EXTENDED BREAK CHECKING*

**Registers at call:**
AH = 33h
AL = subfunction
  00h get current extended break state
  01h set state
    DL = 00h off, 01h on

**Return Registers:**
DL = current state, 00h = off, 01h = on

**Details:** When extended checking is enabled, DOS tests for Control-C and Control-Break on all DOS calls; when disabled, DOS only checks on character I/O function calls.

Under DOS 3.1+ and DR-DOS, this function does not use any of the DOS-internal and may thus be called at any time.

**See also:** Function 3302h

### INT 21h Function 3302h

*DOS 3.x+ internal*

### *GET AND SET EXTENDED CONTROL-BREAK CHECKING STATE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 3302h | DL = old state of extended Break checking |
| DL = new state<br>00h off, 01h on | |

**Details:** This function does not use any of the DOS-internal stacks and may thus be called at any time; one possible use is modifying Control-Break checking from within an interrupt handler or TSR.

This function is not supported by DR-DOS through at least version 6.0.

**See also:** Functions 3300h and 3301h

### INT 21h Function 3305h

*DOS 4+*

### *GET BOOT DRIVE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 3305h | DL = boot drive (1=A:,...) |

**Details:** This function does not use any of the DOS-internal stacks and may thus be called at any time.

NEC 9800-series PCs always call the boot drive A: and assign the other drive letters sequentially to the other drives in the system.

### INT 21h Function 3306h

*DOS 5+*

### *GET TRUE VERSION NUMBER*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 3306h | BL = major version<br>BH = minor version<br>DL = revision (bits 2-0, all others 0)<br>DH = version flags<br>bit 3: DOS is in ROM<br>bit 4: DOS is in HMA |

**Details:** This function always returns the true version number, unlike Function 30h, whose return value may be changed with SETVER.

Because of the conflict from the CBIS PowerLAN redirector, programs should check whether BH is less than 100 (64h) and BL is at least 5 before accepting the returned BX as the true version number; however, even this is not entirely reliable when that redirector is loaded.

This function does not use any of the DOS-internal stacks and may thus be called at any time.

OS/2 v2.1 returns BX=0A14h (version 20.10); the Windows NT DOS box returns BX=3205h (version 5.50)

**BUG:** DR-DOS 5.0 and 6.0 return CF set and AX=0001h for INT 21h Function 33h subfunctions other than 00h-02h and 05h, while MS-DOS returns AL=FFh for invalid subfunctions.

**Conflicts:** CBIS PowerLAN (NI)

**See also:** Function 30h, INT 2Fh Function 122Fh (below)

### INT 21h Function 34h

*DOS 2+*

### *GET ADDRESS OF InDOS FLAG*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 34h | ES:BX -> one-byte InDOS flag |

**Details:** The value of InDOS is incremented whenever an INT 21h function begins and decremented whenever one completes. Although one normally can not call DOS while InDOS is nonzero, during an INT 28h call it is safe to call INT 21h functions higher than 0Ch even though InDOS may be 01h instead of zero.

InDOS alone is not sufficient for determining when it is safe to enter DOS, as the critical error handling decrements InDOS and increments the critical error flag for the duration of the critical error. Thus, it is possible for InDOS to be zero even if DOS is busy.

The critical error flag is the byte immediately following InDOS in DOS 2.x, and the byte BEFORE the InDOS flag in DOS 3+ and DR-DOS 3.41+ (except COMPAQ DOS 3.0, where the critical error flag is located 1AAh bytes BEFORE the critical section flag). For DOS 3.1+, an undocumented call exists to get the address of the critical error flag (see Function 5D06h).

This function was undocumented prior to the release of DOS 5.0.

**See also:** Functions 5D06h and 5D0Bh, INT 28h, DESQview INT 15h Function DE1Fh (Chapter 44)

---

### INT 21h Function 35h

*DOS 2+*

### *GET INTERRUPT VECTOR*

**Registers at call:**
AH = 35h
AL = interrupt number

**Return Registers:**
ES:BX -> current interrupt handler

**Details:** Under DR-DOS 5.0+, this function does not use any of the DOS-internal stacks and may thus be called at any time.

**Conflicts:** FlashTek X-32VM (Chapter 33), Agiplan virus (Chapter 59)

**See also:** Function 25h, Phar Lap Function 2503h (Chapter 33)

---

### INT 21h Function 36h

*DOS 2+*

### *GET FREE DISK SPACE*

**Registers at call:**
AH = 36h
DL = drive number (00h = default, 01h = A:, etc.)

**Return Registers:**
AX = FFFFh if invalid drive
else
  AX = sectors per cluster
  BX = number of free clusters
  CX = bytes per sector
  DX = total clusters on drive

**Details:** The free space on the drive in bytes is AX*BX*CX. The total space on the drive in bytes is AX*CX*DX.

"Lost clusters" are considered to be in use.

According to Dave Williams' MS-DOS reference, the value in DX is incorrect for non-default drives after ASSIGN is run.

**See also:** Functions 1Bh and 1Ch

---

### INT 21h Functions 3700h and 3701h

*DOS 2+*

### *"SWITCHAR" - GET/SET SWITCH CHARACTER*

**Purpose:** Determine or specify the character which is used to introduce commandline switches. This setting is ignored by DOS commands in version 4.0 and higher, but is honored by many third-party programs.

**Registers at call:**
AX = function
  3700h get switch character
  3701h set switch character
    DL = new switch character

**Return Registers:**
AL = status
  00h successful
    DL = current switch character (Function 3700h only)
  FFh unsupported subfunction

**Details:** These functions are documented in some OEM versions of some releases of DOS. They are supported by the OS/2 compatibility box.

Function 3700h always returns DL=2Fh and Function 3701h is ignored for MS-DOS 5+ and DR-DOS 3.41+

## INT 21h Function 37h

*DOS 2.x and 3.3+ only*

### *"AVAILDEV" - SPECIFY \DEV\ PREFIX USE*

**Registers at call:**

AH = 37h
AL = subfunction
    02h get availdev flag
    03h set availdev flag
        DL = new state
            00h \DEV\ is mandatory
            nonzero \DEV\ is optional

**Return Registers:**

AL = status
    00h successful
    FFh unsupported subfunction
DL = state
    00h \DEV\ must precede character device names
    nonzero \DEV\ is optional

**Details:** All versions of DOS from 2.00 allow \DEV\ to be prepended to device names without generating an error even if the directory \DEV does not actually exist (other paths generate an error if they do not exist).

Although MS-DOS 3.3+ and DR-DOS 3.41+ accept these calls, they have no effect, and AL=02h always returns DL=FFh.

## INT 21h Function 38h

*DOS 2+*

### *GET COUNTRY-SPECIFIC INFORMATION*

**Registers at call:**

AH = 38h
DS:DX -> buffer for returned info
  (see Tables 24-16 and 24-17)
AL = 00h get info for current country
---**DOS 3+ only**---
AL = 01h through 0FEh for specific
  country with code <255
AL = 0FFh for specific country with
  code >= 255
    BX = 16-bit country code (see
      Table 24-18)

**Return Registers:**

CF set on error
    AX = error code (02h)
CF clear if successful
    AX = country code (MS-DOS 2.11 only)
    BX = country code (DOS 3+)
    buffer at DS:DX filled

**Details:** This function is not supported by the Borland DPMI host, but no error is returned; as a workaround, one should allocate a buffer in conventional memory with INT 31h Function 0100h and simulate an INT 21h with INT 31h Function 0300h (see Chapter 30).

**See also:** Function 65h, AX PC INT 10h Function 5001h (Chapter 6), NLSFUNC INT 2Fh Function 1404h (Chapter 27), Network Redirector INT 2Fh Function 110Ch (NI)

**24-16 Format of PC-DOS 2.x country info:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | date format 0 = USA mm dd yy<br>1 = Europe dd mm yy<br>2 = Japan yy mm dd |
| 02h | BYTE | currency symbol |
| 03h | BYTE | 00h |
| 04h | BYTE | thousands separator char |
| 05h | BYTE | 00h |
| 06h | BYTE | decimal separator char |
| 07h | BYTE | 00h |
| 08h | 24 BYTEs | reserved |

**24-17 Format of MS-DOS 2.x and DOS 3+ country info:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | date format (see Table 24-16) |
| 02h | 5 BYTEs | ASCIZ currency symbol string |
| 07h | 2 BYTEs | ASCIZ thousands separator |
| 09h | 2 BYTEs | ASCIZ decimal separator |
| 0Bh | 2 BYTEs | ASCIZ date separator |
| 0Dh | 2 BYTEs | ASCIZ time separator |
| 0Fh | BYTE | currency format<br>bit 2 = set if currency symbol replaces decimal point<br>bit 1 = number of spaces between value and currency symbol<br>bit 0 = 0 if currency symbol precedes value<br>1 if currency symbol follows value |
| 10h | BYTE | number of digits after decimal in currency |
| 11h | BYTE | time format<br>bit 0 = 0 if 12-hour clock<br>1 if 24-hour clock |
| 12h | DWORD | address of case map routine<br>(FAR CALL, AL = character to map to upper case [>= 80h]) |
| 16h | 2 BYTEs | ASCIZ data-list separator |
| 18h | 10 BYTEs | reserved |

**24-18 Values for country code:**

| | |
|---|---|
| 001h | United States |
| 002h | Canadian-French |
| 003h | Latin America |
| 01Fh | Netherlands |
| 020h | Belgium |
| 021h | France |
| 022h | Spain |
| 024h | Hungary (not supported by DR-DOS 5.0) |
| 026h | Yugoslavia (not supported by DR-DOS 5.0) |
| 027h | Italy |
| 029h | Switzerland |
| 02Ah | Czechoslovakia/Tjekia (not supported by DR-DOS 5.0) |
| 02Bh | Austria (DR-DOS 5.0) |
| 02Ch | United Kingdom |
| 02Dh | Denmark |
| 02Eh | Sweden |
| 02Fh | Norway |
| 030h | Poland (not supported by DR-DOS 5.0) |
| 031h | Germany |
| 037h | Brazil (not supported by DR-DOS 5.0) |
| 03Dh | International English [Australia in DR-DOS 5.0] |
| 051h | Japan (DR-DOS 5.0, MS-DOS 5.0+) |
| 052h | Korea (DR-DOS 5.0) |
| 056h | China (MS-DOS 5.0+) |
| 058h | Taiwan (MS-DOS 5.0+) |
| 05Ah | Turkey (MS-DOS 5.0+) |
| 15Fh | Portugal |

## INT 21h Function 38h

| | |
|---|---|
| 162h | Iceland |
| 166h | Finland |
| 311h | Middle East/Saudi Arabia (DR-DOS 5.0,MS-DOS 5.0+) |
| 3CCh | Israel (DR-DOS 5.0,MS-DOS 5.0+) |

---

### INT 21h Function 38h, Subfunction FFFFh DOS 3+

***SET COUNTRY CODE***

**Registers at call:**
AH = 38h
DX = FFFFh
AL = 01h through FEh for
specific country with code <255
AL = FFh for specific country with
code >= 255
BX = 16-bit country code (see
Table 24-18)

**Return Registers:**
CF set on error
AX = error code (see Table 24-2)
CF clear if successful

**Details:** This function is not supported by OS/2.

**See also:** NLSFUNC INT 2Fh Function 1403h (Chapter 27)

---

### INT 21h Function 39h DOS 2+

***"MKDIR" - CREATE SUBDIRECTORY***

**Registers at call:**
AH = 39h
DS:DX -> ASCIZ pathname

**Return Registers:**
CF clear if successful
AX destroyed
CF set on error
AX = error code (03h,05h) (see Table 24-2)

**Details:** All directories in the given path except the last must exist.

This function fails if the parent directory is the root and is full.

DOS 2.x-3.3 allow the creation of a directory sufficiently deep that it is not possible to make that directory the current directory because the path would exceed 64 characters.

Under the FlashTek X-32 DOS extender, the pointer is in DS:EDX.

**See also:** Functions 3Ah, 3Bh, and 6Dh; Novell NetWare Function E2h Subfunction 0Ah (NI), Network Redirector INT 2Fh Function 1103h (NI)

---

### INT 21h Function 3Ah DOS 2+

***"RMDIR" - REMOVE SUBDIRECTORY***

**Registers at call:**
AH = 3Ah
DS:DX -> ASCIZ pathname of
directory to be removed

**Return Registers:**
CF clear if successful
AX destroyed
CF set on error
AX = error code (03h,05h,06h,10h) (see Table 24-2)

**Details:** The directory must be empty (contain only '.' and '..' entries).

Under the FlashTek X-32 DOS extender, the pointer is in DS:EDX.

**See also:** Functions 39h and 3Bh, Novell NetWare Function E2h Subfunction 0Bh (NI), Network Redirector INT 2Fh Function 1101h (NI)

---

### INT 21h Function 3Bh DOS 2+

***"CHDIR" - SET CURRENT DIRECTORY***

**Registers at call:**
AH = 3Bh
DS:DX -> ASCIZ pathname to become
current directory (max 64 bytes)

**Return Registers:**
CF clear if successful
AX destroyed
CF set on error
AX = error code (03h) (see Table 24-2)

**Details:** If the new directory name includes a drive letter, the default drive is not changed, only the current directory on that drive.

Changing the current directory also changes the directory in which FCB file calls operate.

Under the FlashTek X-32 DOS extender, the pointer is in DS:EDX.

**See also:** Function 47h, Network Redirector INT 2Fh Function 1105h (NI)

---

### INT 21h Function 3Ch

*DOS 2+*

### *"CREAT" - CREATE OR TRUNCATE FILE*

**Registers at call:**
AH = 3Ch
CX = file attributes (see Table 24-21, "directory" bit must be clear)
DS:DX -> ASCIZ filename

**Return Registers:**
CF clear if successful
AX = file handle
CF set on error
AX = error code (03h,04h,05h) (see Table 24-2)

**Details:** If a file with the given name exists, it is truncated to zero length.

Under the FlashTek X-32 DOS extender, the pointer is in DS:EDX.

DR-DOS checks the system password or explicitly supplied password at the end of the filename against the reserved field in the directory entry before allowing access.

**See also:** Functions 16h, 3Dh, 5Ah, and 5Bh; European MS-DOS Function 93h (Chapter 25), Network Redirector INT 2Fh Function 1117h (NI)

---

### INT 21h Function 3Dh

*DOS 2+*

### *"OPEN" - OPEN EXISTING FILE*

**Registers at call:**
AH = 3Dh
AL = access and sharing modes (see Table 24-19)
DS:DX -> ASCIZ filename
CL = attribute mask of files to look for (server call only)

**Return Registers:**
CF clear if successful
AX = file handle
CF set on error
AX = error code (01h,02h,03h,04h,05h,0Ch,56h) (see Table 24-2)

**Details:** The file pointer is set to the start of the file.

File handles which are inherited from a parent also inherit sharing and access restrictions.

Files may be opened even if given the hidden or system attributes.

Under the FlashTek X-32 DOS extender, the pointer is in DS:EDX.

DR-DOS checks the system password or explicitly supplied password at the end of the filename against the reserved field in the directory entry before allowing access.

Sharing modes are effective on local drives only if SHARE is loaded.

**Conflicts:** JD-448 virus (Chapter 59)

**See also:** Functions 0Fh, 3Ch, 4301h, and 5D00h; INT 2Fh Function 1226h (below); Network Redirector INT 2Fh Function 1116h (NI)

#### 24-19 Bitfields for access and sharing modes:

| | |
|---|---|
| bits 2-0 | access mode |
| | 000 read only |
| | 001 write only |
| | 010 read/write |
| | 011 (DOS 5+ internal) passed to redirector on EXEC to allow case-sensitive filenames |
| bit 3 | reserved (0) |
| bits 6-4 | sharing mode (DOS 3+) |
| | 000 compatibility mode |
| | 001 "DENYALL" prohibit both read and write access by others |
| | 010 "DENYWRITE" prohibit write access by others |
| | 011 "DENYREAD" prohibit read access by others |

100 "DENYNONE" allow full access by others
111 network FCB (only available during server call)
bit 7 inheritance
if set, file is private to current process and will not be inherited by child processes

**24-20 File sharing behavior:**

| First Open | | Second and subsequent Opens | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Compatib. | | | Deny All | | | Deny Write | | | Deny Read | | | Deny None | | | | | |
| | | R | W | RW | R | W | RW | R | W | RW | R | W | RW | R | W | RW | R | W | RW |
| Compat | R | Y | Y | Y | N | N | N | 1 | N | N | N | N | N | 1 | N | N | | | |
| | W | Y | Y | Y | N | N | N | N | N | N | N | N | N | N | N | N | | | |
| | RW | Y | Y | Y | N | N | N | N | N | N | N | N | N | N | N | N | | | |
| Deny All | R | C | C | C | N | N | N | N | N | N | N | N | N | N | N | N | | | |
| | W | C | C | C | N | N | N | N | N | N | N | N | N | N | N | N | | | |
| | RW | C | C | C | N | N | N | N | N | N | N | N | N | N | N | N | | | |
| Deny Write | R | 2 | C | C | N | N | N | Y | N | N | N | N | N | Y | N | N | | | |
| | W | C | C | C | N | N | N | N | N | N | Y | N | N | Y | N | N | | | |
| | RW | C | C | C | N | N | N | N | N | N | N | N | N | Y | N | N | | | |
| Deny Read | R | C | C | C | N | N | N | N | Y | N | N | N | N | N | Y | N | | | |
| | W | C | C | C | N | N | N | N | N | N | N | Y | N | N | Y | N | | | |
| | RW | C | C | C | N | N | N | N | N | N | N | N | N | N | Y | N | | | |
| Deny None | R | 2 | C | C | N | N | N | Y | Y | Y | N | N | N | Y | Y | Y | | | |
| | W | C | C | C | N | N | N | N | N | N | Y | Y | Y | Y | Y | Y | | | |
| | RW | C | C | C | N | N | N | N | N | N | N | N | N | Y | Y | Y | | | |

Legend: Y = open succeeds
N = open fails with error code 05h
C = open fails, INT 24h generated
1 = open succeeds if file read-only, else fails with error code
2 = open succeeds if file read-only, else fails with INT 24h

## INT 21h Function 3Eh

DOS 2+

### *"CLOSE" - CLOSE FILE*

| Registers at call: | Return Registers: |
|---|---|
| AH = 3Eh | CF clear if successful |
| BX = file handle | AX destroyed |
| | CF set on error |
| | AX = error code (06h) (see Table 24-2) |

**Details:** If the file was written to, any pending disk writes are performed, the time and date stamps are set to the current time, and the directory entry is updated.

**See also:** Functions 10h, 3Ch, and 3Dh; INT 2Fh Function 1227h (below); Network Redirector INT 2Fh Function 1106h (NI)

---

DOS 2+

### INT 21h Function 3Fh
### *"READ" - READ FROM FILE OR DEVICE*

**Registers at call:**
AH = 3Fh
BX = file handle
CX = number of bytes to read
DS:DX -> buffer for data

**Return Registers:**
CF clear if successful
  AX = number of bytes actually read (0 if at EOF before call)
CF set on error
  AX = error code (05h,06h) (see Table 24-2)

**Details:** Data is read beginning at the current file position, and the file position is updated after a successful read.

The returned AX may be smaller than the request in CX if a partial read occurred. If reading from CON, the read stops at the first carriage return.

Under the FlashTek X-32 DOS extender, the pointer is in DS:EDX.

**See also:** Functions 27h and 40h, European MS-DOS Function 93h (Chapter 25), INT 2Fh Function 1229h (below), Network Redirector INT 2Fh Function 1108h (NI)

---

DOS 2+

### INT 21h Function 40h
### *"WRITE" - WRITE TO FILE OR DEVICE*

**Registers at call:**
AH = 40h
BX = file handle
CX = number of bytes to write
DS:DX -> data to write

**Return Registers:**
CF clear if successful
  AX = number of bytes actually written
CF set on error
  AX = error code (05h,06h) (see Table 24-2)

**Details:** If CX is zero, no data is written, and the file is truncated or extended to the current position.

Data is written beginning at the current file position, and the file position is updated after a successful write.

The usual cause for AX < CX on return is a full disk.

**BUG:** A write of zero bytes will appear to succeed when it actually failed if the write is extending the file and there is not enough disk space for the expanded file (DOS 5.0-6.0); one should therefore check whether the file was in fact extended by seeking to 0 bytes from the end of the file (INT 21h Function 4202h with CX=DX=0).

Under the FlashTek X-32 DOS extender, the pointer is in DS:EDX.

**Conflicts:** FARTBELL (Chapter 22)

**See also:** Functions 28h and 3Fh, European MS-DOS Function 93h (Chapter 25), Network Redirector INT 2Fh Function 1109h (NI)

---

DOS 2+

### INT 21h Function 41h
### *"UNLINK" - DELETE FILE*

**Registers at call:**
AH = 41h
DS:DX -> ASCIZ filename (no wildcards, but see below)
CL = attribute mask for deletion (server call only, see below)

**Return Registers:**
CF clear if successful
  AX destroyed (DOS 3.3) AL seems to be drive of deleted file
CF set on error
  AX = error code (02h,03h,05h) (see Table 24-2)

**Details:** Under DOS 3.1+, wildcards are allowed if this function is invoked via Function 5D00h (Server Call), in which case the filespec must be canonical (as returned by Function 60h), and only files matching the attribute mask in CL are deleted.

DR-DOS 5.0-6.0 returns error code 03h if invoked via Function 5D00h.

DOS does not erase the file's data; it merely becomes inaccessible because the FAT chain for the file is cleared.

Deleting a file which is currently open may lead to filesystem corruption. Unless SHARE is loaded, DOS does not close the handles referencing the deleted file, thus allowing writes to a non-existent file. Under DRDOS and DR Multiuser DOS, this function will fail if the file is currently open.

Under the FlashTek X-32 DOS extender, the pointer is in DS:EDX

**BUG:** DR-DOS 3.41 crashes if called via Function 5D00h.

**Conflicts:** SoftLogic Data Guardian (Chapter 66)

**See also:** Functions 13h, 4301h, 5D00h, and 60h; Novell DOS Function 4380h (Chapter 26); Novell NetWare Function F244h (NI); Network Redirector INT 2Fh Function 1113h (NI)

### INT 21h Function 42h

*DOS 2+*

### *"LSEEK" - SET CURRENT FILE POSITION*

**Registers at call:**

AH = 42h

AL = origin of move

- 00h start of file
- 01h current file position
- 02h end of file

BX = file handle

CX:DX = offset from origin of new file position

**Return Registers:**

CF clear if successful

DX:AX = new file position in bytes from start of file

CF set on error

AX = error code (01h,06h) (see Table 24-2)

**Details:** For origins 01h and 02h, the pointer may be positioned before the start of the file; no error is returned in that case, but subsequent attempts at I/O will produce errors.

If the new position is beyond the current end of file, the file will be extended by the next write (see Function 40h).

**BUG:** Using this method to grow a file from zero bytes to a very large size can corrupt the FAT in some versions of DOS; the file should first be grown from zero to one byte and then to the desired large size.

**See also:** Function 24h, INT 2Fh Function 1228h (below)

### INT 21h Function 4300h

*DOS 2+*

### *GET FILE ATTRIBUTES*

**Registers at call:**

AX = 4300h

DS:DX -> ASCIZ filename

**Return Registers:**

CF clear if successful

CX = file attributes (see Table 24-21)

AX = CX (DR-DOS 5.0)

CF set on error

AX = error code (01h,02h,03h,05h) (see Table 24-2)

**Details:** Under the FlashTek X-32 DOS extender, the filename pointer is in DS:EDX.

Under DR-DOS 3.41 and 5.0, attempts to change the subdirectory bit are simply ignored without an error.

**See also:** Function 4301h, Banyan VINES Function 4310h (NI), Novell NetWare Function B6h (NI), Network Redirector INT 2Fh Function 110Fh (NI)

### INT 21h Function 4301h

*DOS 2+*

### *"CHMOD" - SET FILE ATTRIBUTES*

**Registers at call:**

AX = 4301h

CX = new file attributes (see Table 24-21)

DS:DX -> ASCIZ filename

**Return Registers:**

CF clear if successful

AX destroyed

CF set on error

AX = error code (01h,02h,03h,05h) (see Table 24-2)

**Details:** This function will not change the volume label or directory attribute bits, but will change the other attribute bits of a directory (the directory bit must be cleared to successfully change the other attributes of a directory, but the directory will not be changed to a normal file as a result).

MS-DOS 4.01 reportedly closes the file if it is currently open. DOS 5.0 SHARE will close the file if it is currently open in sharing- compatibility mode, otherwise a sharing violation critical error is generated if the file is currently open.

For security reasons, the Novell NetWare execute-only bit can never be cleared; the file must be deleted and recreated.

Under the FlashTek X-32 DOS extender, the filename pointer is in DS:EDX.

**See also:** Function 4300h, Banyan VINES Function 4311h (NI), Network Redirector INT 2Fh Function 110Eh (NI)

### 24-21 Bitfields for file attributes:

| | |
|---|---|
| bit 7 | shareable (Novell NetWare) |
| bit 6 | unused |
| bit 5 | archive |
| bit 4 | directory |
| bit 3 | volume label |
| | execute-only (Novell NetWare) |
| bit 2 | system |
| bit 1 | hidden |
| bit 0 | read-only |

## INT 21h Function 4400h

DOS 2+

### *IOCTL - GET DEVICE INFORMATION*

**Registers at call:**
AX = 4400h
BX = handle

**Return Registers:**
CF clear if successful
DX = device information word (see Table 24-22)
AX destroyed
CF set on error
AX = error code (01h,05h,06h) (see Table 24-2)

**Details:** The value in DH corresponds to the high byte of the device driver's attribute word if the handle refers to a character device.

Novell NetWare reportedly does not return a drive number in bits 5-0 for a disk file.

**See also:** Function 4401h, INT 2Fh Function 122Bh (below)

### 24-22 Bitfields for device information word:

bit 14: device driver can process IOCTL requests (see Function 4402h)
bit 13: output until busy supported
bit 11: driver supports OPEN/CLOSE calls
bit 7: set (indicates device)
bit 6: EOF on input
bit 5: raw (binary) mode
bit 4: device is special (uses INT 29h)
bit 3: clock device
bit 2: NUL device
bit 1: standard output
bit 0: standard input
bit 15: file is remote (DOS 3+)
bit 14: don't set file date/time on closing (DOS 3+)
bit 11: media not removable
bit 8: (DOS 4 only) generate INT 24h if no disk space on write or read past end of file

bit 7: clear (indicates file)
bit 6: file has not been written
bits 5-0: drive number (0 = A:)

---

### INT 21h Function 4401h DOS 2+
### *IOCTL - SET DEVICE INFORMATION*

**Registers at call:**
AX = 4401h
BX = handle (must refer to character device)
DX = device information word (see Table 24-22)
(DH must be zero)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**See also:** Function 4400h, INT 2Fh Function 122Bh (below)

---

### INT 21h Function 4402h DOS 2+
### *IOCTL - READ FROM CHARACTER DEVICE CONTROL CHANNEL*

**Registers at call:**
AX = 4402h
BX = file handle referencing character device
CX = number of bytes to read
DS:DX -> buffer

**Return Registers:**
CF clear if successful
AX = number of bytes actually read
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Details:** The data format is driver-specific (various drivers are covered in other chapters; see the index under INT 21h, Functions 4402h and 4403h).

**See also:** Functions 4400h, 4403h, and 4404h; INT 2Fh Function 122Bh (below)

---

### INT 21h Function 4403h DOS 2+
### *IOCTL - WRITE TO CHARACTER DEVICE CONTROL CHANNEL*

**Registers at call:**
AX = 4403h
BX = file handle referencing character device
CX = number of bytes to write
DS:DX -> data to write

**Return Registers:**
CF clear if successful
AX = number of bytes actually written
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Details:** The data format is driver-specific (various drivers are covered in other chapters; see the index under INT 21h, Functions 4402h and 4403h).

If the file handle refers to "4DOSSTAK", the 4DOS (v2.x-3.03) KEYSTACK.SYS driver will push the specified characters on the keyboard stack; similarly for "NDOSSTAK", the NDOS KEYSTACK.SYS driver will push the characters onto the keyboard stack.

**See also:** Functions 4400h, 4402h, and 4405h; INT 2Fh Function 122Bh (below); 4DOS INT 2Fh Functions D44Dh and D44Fh (Chapter 28)

---

### INT 21h Functions 4404h and 4405h DOS 2+
### *IOCTL - READ/WRITE BLOCK DEVICE CONTROL CHANNEL*

**Registers at call:**
AX = function
4404h read from control channel
4405h write to control channel
BL = drive number (00h = default, 01h = A:, etc.)
CX = number of bytes to read or write
DS:DX -> buffer for/containing data

**Return Registers:**
CF clear if successful
AX = number of bytes actually transferred
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Details:** The data format is driver-specific.
**See also:** Functions 4402h and 4403h, INT 2Fh Function 122Bh (below)

---

## INT 21h Functions 4406h and 4407h
DOS 2+

### *IOCTL - GET INPUT/OUTPUT STATUS*

**Registers at call:**
AX = function
  4406h get input status
  4407h get output status
BX = file handle

**Return Registers:**
CF clear if successful
  AL = input or output status
    00h not ready (device)
      at EOF (file, Function 4406h only)
    FFh ready
CF set on error
  AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Details:** Files may not register as being at EOF for input if positioned there by Function 42h.

For DOS 2+, files are always ready for output, even if the disk is full or no media is in the drive.

**See also:** INT 2Fh Function 122Bh (below)

---

## INT 21h Function 4408h
DOS 3.0+

### *IOCTL - CHECK IF BLOCK DEVICE REMOVABLE*

**Registers at call:**
AX = 4408h
BL = drive number (00h = default, 01h = A:, etc.)

**Return Registers:**
CF clear if successful
  AX = type (0000h removable, 0001h fixed)
CF set on error
  AX = error code (01h,0Fh) (see Table 24-2)

**See also:** Functions 4400h and 4409h, INT 2Fh Function 122Bh (below)

---

## INT 21h Function 4409h
DOS 3.1+

### *IOCTL - CHECK IF BLOCK DEVICE REMOTE*

**Registers at call:**
AX = 4409h
BL = drive number (00h = default, 01h = A:, etc.)

**Return Registers:**
CF clear if successful
  DX = device attribute word
    bit 15: drive is SUBSTituted
    bit 12: drive is remote
    bit 9: direct I/O not allowed
CF set on error
  AX = error code (01h,0Fh) (see Table 24-2)

**Details:** On local drives, DX bits not listed above are the attribute word from the device driver header (see Table 24-83); for remote drives, the other bits appear to be undefined for DOS versions prior to 5.0 (they are all cleared in DOS 5+).

**See also:** Functions 4400h, 4408h, and 440Ah; INT 2Fh Function 122Bh (below)

---

## INT 21h Function 440Ah
DOS 3.1+

### *IOCTL - CHECK IF HANDLE IS REMOTE*

**Registers at call:**
AX = 440Ah
BX = handle

**Return Registers:**
CF clear if successful
  DX = attribute word (as stored in SFT)
    bit 15: set if remote
    bit 14: date/time not set on close
CF set on error
  AX = error code (01h,06h) (see Table 24-2)

**Details:** If the file is remote, Novell Advanced NetWare 2.0 returns the number of the file server on which the handle is located in CX.

DR-DOS 3.41 and 5.0 clear all bits of DX except bit 15.

**See also:** Functions 4400h, 4409h, and 52h; INT 2Fh Function 122Bh (below)

### INT 21h Function 440Bh — *DOS 3.1+*
### *IOCTL - SET SHARING RETRY COUNT*

**Registers at call:**
AX = 440Bh
CX = pause between retries (default 1)
DX = number of retries (default 3)

**Return Registers:**
CF clear if successful
CF set on error
  AX = error code (01h) (see Table 24-2)

**Details:** The delay is dependent on processor speed, since the value in CX specifies the number of 64K-iteration empty loops to execute.

If DX=0000h on entry, the retry count is left unchanged.

**See also:** Function 52h, INT 2Fh Functions 1224h and 122Bh (below)

### INT 21h Function 440Ch — *DOS 3.2+*
### *IOCTL - GENERIC CHARACTER DEVICE REQUEST*

**Registers at call:**
AX = 440Ch
BX = device handle
CH = category code (see Table 24-23)
CL = function (see Table 24-24)
DS:DX -> parameter block (see Tables 24-25 to 24-31)
**---European MS-DOS 4.0 and OS/2 comp. box---**
SI = parameter to pass to driver
DI = parameter to pass to driver

**Return Registers:**
CF set on error
  AX = error code (see Table 24-2)
CF clear if successful
  DS:DX -> iteration count if CL=65h
  SI = returned value (European MS-DOS 4.0, OS/2 comp box)
  DI = returned value (European MS-DOS 4.0, OS/2 comp box)

**Details:** Bit assignments for function code in CL:
bit 7: set to ignore if unsupported, clear to return error
bit 6: set if passed to driver, clear if intercepted by DOS
bit 5: set if queries data from device, clear if sends command
bits 4-0: subfunction.

DR-DOS 3.41 and 5.0 return error code 16h on CL=45h,65h if the device does not support a retry counter.

**See also:** Function 440Dh, INT 2Fh Functions 0802h and 122Bh, DR-DOS INT 2Fh Function 14FFh (Chapter 26), ANSI.SYS INT 2Fh Function 1A01h (Chapter 27)

**24-23 Category codes:**

| | |
|---|---|
| 00h | unknown (DOS 3.3+) |
| 01h | COMn: (DOS 3.3+) |
| 03h | CON (DOS 3.3+) |
| 05h | LPTn: |
| 9Eh | Media Access Control driver (STARLITE) |
| 00h-7Fh | reserved for Microsoft |
| 80h-FFh | reserved for OEM/user-defined |

**24-24 Function codes:**

| | |
|---|---|
| 00h | MAC driver Bind (STARLITE) |
| 45h | set iteration (retry) count |
| 4Ah | select code page |
| 4Ch | start code-page preparation |

| | |
|---|---|
| 4Dh | end code-page preparation |
| 5Fh | set display information (DOS 4+) |
| 65h | get iteration (retry) count |
| 6Ah | query selected code page |
| 6Bh | query prepare list |
| 7Fh | get display information (DOS 4+) |

**24-25 Format of parameter block for function 00h:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 8 BYTEs | ASCIZ signature "STARMAC" |
| 08h | WORD | version |
| 0Ah | WORD | flags |
| | | bit 0: media requires connect or listen request before use |
| | | bit 1: network is a LAN (broadcast/multicast supported) |
| | | bit 2: point-to-point network |
| 0Ch | WORD | handle for use with MAC driver's private interface (filled in by MAC driver) |
| 0Eh | WORD | context |
| 10h | WORD | approximate speed in KB/sec (filled in by MAC driver) |
| 12h | WORD | approximate cost in cents per hour (filled in by MAC driver) |
| 14h | WORD | maximum packet size in bytes (filled in by MAC driver) |
| 16h | WORD | addressing format (filled in by MAC driver) |
| | | 0000h general addressing |
| | | 0001h Ethernet addressing |
| | | 0002h Token Ring addressing |
| | | 0003h Token Bus addressing |
| 18h | DWORD | Send entry point (filled in by MAC driver) |
| 1Ch | DWORD | RegisterEventHandler entry point (filled in by MAC driver) |
| 20h | DWORD | SetPacketFilter entry point (filled in by MAC driver) |
| 24h | DWORD | UnBind entry point (filled in by MAC driver) |

**24-26 Format of parameter block for function 45h:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of times output is attempted before driver assumes device is busy |

**24-27 Format of parameter block for functions 4Ah and 6Ah:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | length of data |
| 02h | WORD | code page ID |
| 04h | 2N BYTEs | DCBS (double byte character set) lead byte range start/end for each of N ranges (DOS 4.0) |
| | WORD | 0000h end of data (DOS 4.0) |

**24-28 Format of parameter block for function 4Ch:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | flags |
| | | DISPLAY.SYS = 0000h |
| | | PRINTER.SYS bit 0 clear to prepare downloaded font, set to prepare cartridge selection |
| 02h | WORD | length of remainder of parameter block |
| 04h | WORD | number of code pages following |
| 06h | N WORDs | code page 1,...,N |

## INT 21h Function 440Ch

### 24-29 Format of parameter block for function 4Dh:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | length of data |
| 02h | WORD | code page ID |

### 24-30 Format of parameter block for functions 5Fh and 7Fh:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | level (0 for DOS 4.x-6.0) |
| 01h | BYTE | reserved (0) |
| 02h | WORD | length of following data (14) |
| 04h | WORD | control flags<br>bit 0 set for blink, clear for intensity<br>bits 1 to 15 reserved |
| 06h | BYTE | mode type (1=text, 2=graphics) |
| 07h | BYTE | reserved (0) |
| 08h | WORD | colors, N bits per pixel (0 = monochrome) |
| 0Ah | WORD | pixel columns |
| 0Ch | WORD | pixel rows |
| 0Eh | WORD | character columns |
| 10h | WORD | character rows |

### 24-31 Format of parameter block for function 6Bh:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | length of following data |
| 02h | WORD | number of hardware code pages |
| 04h | N WORDs | hardware code pages 1,...,N |
|  | WORD | number of prepared code pages |
|  | N WORDs | prepared code pages 1,...,N |

---

## INT 21h Function 440Dh

*DOS 3.2+*

### *IOCTL - GENERIC BLOCK DEVICE REQUEST*

**Registers at call:**

AX = 440Dh
BL = drive number (00h=default,01h=A:,etc.)
CH = category code
  08h disk drive
  00h-7Fh reserved for Microsoft
  80h-FFh reserved for OEM/user-defined
CL = minor code (function) (see Table 24-32)
DS:DX -> parameter block (see Tables 24-33 to 24-40)
SI:DI = driver data (OS/2 compatibility box)

**Return Registers:**

CF set on error
  AX = error code (01h,02h) (see Table 24-2)
CF clear if successful
  DS:DX -> data block if CL=60h or CL=61h

**Details:** DOS 4.01 seems to ignore the high byte of the number of directory entries in the BPB for diskettes.

Functions 46h and 66h were undocumented in DOS 4.x, documented for DOS 5+.

The DUBLDISK.SYS v2.6 driver only supports minor codes 60h and 67h.

DR-DOS 3.41-6.0 only support minor codes 40h-42h and 60h-62h; all other minor codes return error code 16h.

**See also:** Functions 440Ch and 69h; INT 2Fh Functions 0802h and 122Bh (below)

**24-32 Values for minor code:**

| | |
|---|---|
| 00h | (OS/2) \ used to lock/unlock a drive |
| 01h | (OS/2) / |
| 40h | set device parameters |
| 41h | write logical device track |
| 42h | format and verify logical device track |
| 46h | (DOS 4+) set volume serial number (see also Function 69h) |
| 47h | (DOS 4+) set access flag |
| 60h | get device parameters |
| 61h | read logical device track |
| 62h | verify logical device track |
| 66h | (DOS 4+) get volume serial number (see also Function 69h) |
| 67h | (DOS 4+) get access flag |
| 68h | (DOS 5+) sense media type |

**24-33 Format of parameter block for functions 40h, 60h:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | special functions<br>bit 0 set if function to use current BPB, clear if Device BIOS Parameter Block field contains new default BPB<br>bit 1 set if function to use track layout fields only; must be clear if CL=60h<br>bit 2 set if all sectors in track same size (should be set)<br>bits 3-7 reserved |
| 01h | BYTE | device type (see Table 24-34) |
| 02h | WORD | device attributes<br>bit 0 set if nonremovable medium<br>bit 1 set if door lock ("changeline") supported<br>bits 2-15 reserved |
| 04h | WORD | number of cylinders |
| 06h | BYTE | media type<br>for 1.2M drive<br>00h 1.2M disk (default)<br>01h 320K/360K disk<br>F8h for DUBLDISK.SYS v2.6 expanded drives<br>always 00h for other drive types |
| 07h | 31 BYTEs | device BPB (see Table 24-95), bytes after BPB offset 1Eh omitted |
| **---function 40h only---** | | |
| 26h | WORD | number of sectors per track (start of track layout field), max 63 |
| 28h | N word | pairs: number, size of each sector in track |

**24-34 Values for device type:**

| | |
|---|---|
| 00h | 320K/360K disk |
| 01h | 1.2M disk |
| 02h | 720K disk |
| 03h | single-density 8-inch disk |
| 04h | double-density 8-inch disk |
| 05h | fixed disk |
| 06h | tape drive |
| 07h | (DOS 3.3+) other type of block device, normally 1.44M floppy |
| 08h | read/write optical disk |
| 09h | (DOS 5+) 2.88M floppy |

### 24-35 Format of parameter block for functions 41h, 61h:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | special functions (reserved, must be zero) |
| 01h | WORD | number of disk head |
| 03h | WORD | number of disk cylinder |
| 05h | WORD | number of first sector to read/write |
| 07h | WORD | number of sectors |
| 09h | DWORD | transfer address |

### 24-36 Format of parameter block for function 42h:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | reserved, must be zero (DOS <3.2)<br>bit 0 = 0: format/verify track<br>1: format status call (DOS 3.2+), don't actually format<br>bits 1-7 reserved, must be zero<br>on return (DOS 4.x):<br>bit 0: set if specified tracks, sectors/track supported<br>bit 1: set if function not supported by BIOS<br>bit 2: set if specified tracks, sectors/track not supported<br>bit 3: set if no disk in drive |
| 01h | WORD | number of disk head |
| 03h | WORD | number of disk cylinder |

### 24-37 Format of parameter block for function 62h:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | reserved, must be zero (DOS <3.2)<br>bit 0 = 0: verify single track<br>1: verify multiple tracks<br>bits 1-7 reserved, must be zero<br>on return (DOS 4.x):<br>bit 0: set if specified tracks, sectors/track supported<br>bit 1: set if function not supported by BIOS<br>bit 2: set if specified tracks, sectors/track not supported<br>bit 3: set if no disk in drive |
| 01h | WORD | number of disk head |
| 03h | WORD | number of disk cylinder |
| 05h | WORD | number of tracks to verify (equivalent to 255 or fewer sectors) |

### 24-38 Format of parameter block for functions 46h, 66h:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | (call) info level (should be 0000h) |
| 02h | DWORD | disk serial number (binary) |
| 06h | 11 BYTEs | volume label or "NO NAME " |
| 11h | 8 BYTEs | filesystem type "FAT12 " or "FAT16 " (CL=66h only) |

### 24-39 Format of parameter block for functions 47h, 67h:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | special-function field (must be zero) |
| 01h | BYTE | disk-access flag, nonzero if access allowed by driver |

### 24-40 Format of parameter block for function 68h:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | 01h for default media type, 00h for any other<br>(see also INT 13h Function 20h, Chapter 10) |
| 01h | BYTE | 02h for 720K, 07h for 1.44M, 09h for 2.88M |

---

### INT 21h Function 440Eh

DOS 3.2+

### *IOCTL - GET LOGICAL DRIVE MAP*

**Registers at call:**
AX = 440Eh
BL = drive number
(00h=default,01h=A:,etc)

**Return Registers:**
CF set on error
AX = error code (01h,0Fh) (see Table 24-2)
CF clear if successful
AL = 00h block device has only one logical drive assigned
1..26 the last letter used to reference the drive (1=A:,etc)

**Details:** DR-DOS 3.41-5.0 DRIVER.SYS does not support drive mapping and thus always returns AL=00h.

**See also:** Function 440Fh, INT 2Fh Function 122Bh (below)

---

### INT 21h Function 440Fh

DOS 3.2+

### *IOCTL - SET LOGICAL DRIVE MAP*

**Registers at call:**
AX = 440Fh
BL = physical drive number
(00h=default,01h=A:,etc))

**Return Registers:**
CF set on error
AX = error code (01h,0Fh) (see Table 24-2)
CF clear if successful
drive now responds to next logical drive number

**Details:** This function changes the mapping of logical drives to physical drives, similar to DOS's treatment of a single physical floppy drive as both A: and B:

DR-DOS 3.41-5.0 DRIVER.SYS does not support drive mapping and thus always returns an error on this function.

**See also:** Function 440Eh, INT 2Fh Function 122Bh (below)

---

### INT 21h Function 4410h

DOS 5+

### *IOCTL - QUERY GENERIC IOCTL CAPABILITY (HANDLE)*

**Registers at call:**
AX = 4410h
BX = handle for device
CH = category code (see Table 24-23)
CL = function code

**Return Registers:**
CF clear if successful
AX = 0000h specified IOCTL function is supported
CF set on error
AL = 01h IOCTL capability not available

**Details:** A program which wishes to use Generic IOCTL calls beyond those in the standard DOS 3.2 set may use this call first to see whether a particular call is supported.

**Conflicts:** NewSpace (Chapter 16)

**See also:** Functions 440Ch, 440Dh, and 4411h

---

### INT 21h Function 4411h

DOS 5+

### *IOCTL - QUERY GENERIC IOCTL CAPABILITY (DRIVE)*

**Registers at call:**
AX = 4411h
BL = drive number
CH = category code
(see Function 440Dh)
CL = minor (function) code
(see Table 24-32)

**Return Registers:**
CF clear if successful
AX = 0000h specified IOCTL function is supported
CF set on error
AL = 01h IOCTL capability not available

**Details:** A program which wishes to use Generic IOCTL calls beyond those in the standard DOS 3.2 set may use this call first to see whether a particular call is supported.

**Conflicts:** NewSpace (Chapter 16)

**See also:** Functions 440Ch, 440Dh, and 4410h

### INT 21h Function 440Eh

---

### INT 21h Function 45h
DOS 2+

### "DUP" - DUPLICATE FILE HANDLE

| Registers at call: | Return Registers: |
|---|---|
| AH = 45h | CF clear if successful |
| BX = file handle | AX = new handle |
| | CF set on error |
| | AX = error code (04h,06h) (see Table 24-2) |

**Details:** Moving the file pointer for either handle will also move it for the other, because both will refer to the same system file table.

For DOS versions prior to 3.3, file writes may be forced to disk by duplicating the file handle and closing the duplicate. DOS 3.3+ provides Function 68h to force an immediate disk update.

**See also:** Functions 3Dh and 46h

---

### INT 21h Function 46h
DOS 2+

### "DUP2", "FORCEDUP" - FORCE DUPLICATE FILE HANDLE

| Registers at call: | Return Registers: |
|---|---|
| AH = 46h | CF clear if successful |
| BX = file handle | CF set on error |
| CX = file handle to become duplicate of first handle | AX = error code (04h,06h) (see Table 24-2) |

**BUG:** DOS 3.30 hangs if BX=CX on entry.

**Details:** This function closes the file with handle CX if it is still open.

Moving the file pointer for either handle will also move it for the other, because both will refer to the same system file table.

**See also:** Functions 3Dh and 45h

---

### INT 21h Function 47h
DOS 2+

### "CWD" - GET CURRENT DIRECTORY

| Registers at call: | Return Registers: |
|---|---|
| AH = 47h | CF clear if successful |
| DL = drive number (00h = default, 01h = A:, etc.) | AX = 0100h (undocumented) |
| | CF set on error |
| DS:SI -> 64-byte buffer for ASCIZ pathname | AX = error code (0Fh) (see Table 24-2) |

**Details:** The returned path does not include a drive or the initial backslash.

Many Microsoft products for Windows rely on AX being 0100h on success.

Under the FlashTek X-32 DOS extender, the buffer pointer is in DS:ESI.

**See also:** Functions 19h and 3Bh, DESQview INT 15h Function DE25h (Chapter 44)

---

### INT 21h Function 48h
DOS 2+

### ALLOCATE MEMORY

| Registers at call: | Return Registers: |
|---|---|
| AH = 48h | CF clear if successful |
| BX = number of paragraphs to allocate | AX = segment of allocated block |
| | CF set on error |
| | AX = error code (07h,08h) (see Table 24-2) |
| | BX = size of largest available block |

**Details:** DOS 2.1-6.0 coalesces free blocks while scanning for a block to allocate.

.COM programs are initially allocated the largest available memory block, and should free some memory with Function 49h before attempting any allocations.

Under the FlashTek X-32 DOS extender, EBX contains a protected-mode near pointer to the allocated block on a successful return.

**See also:** Functions 49h, 4Ah, and 58h; European MS-DOS Function 83h (Chapter 25)

---

## INT 21h Function 49h

DOS 2+

### *FREE MEMORY*

**Registers at call:**
AH = 49h
ES = segment of block to free

**Return Registers:**
CF clear if successful
CF set on error
  AX = error code (07h,09h) (see Table 24-2)

**Details:** This function apparently never returns an error 07h, despite official documents; DOS 3.30 code contains only an error 09h exit.

DOS 2.1-6.0 does not coalesce adjacent free blocks when a block is freed, only when a block is allocated or resized.

The code for this function is identical in DOS 2.1-6.0 except for calls to start/end a critical section in DOS 3+.

**See also:** Functions 48h and 4Ah

---

## INT 21h Function 4Ah

DOS 2+

### *RESIZE MEMORY BLOCK*

**Registers at call:**
AH = 4Ah
BX = new size in paragraphs
ES = segment of block to resize

**Return Registers:**
CF clear if successful
  AX destroyed
CF set on error
  AX = error code (07h,08h,09h) (see Table 24-2)
  BX = maximum paragraphs available for specified memory block

**Details:** Under DOS 2.1-6.0, if there is insufficient memory to expand the block as much as requested, the block will be made as large as possible.

DOS 2.1-6.0 coalesces any free blocks immediately following the block to be resized.

**See also:** Functions 48h and 49h, European MS-DOS Function 83h (Chapter 25)

---

## INT 21h Function 4Bh

DOS 2+

### *"EXEC" - LOAD AND/OR EXECUTE PROGRAM*

**Registers at call:**
AH = 4Bh
AL = type of load
  00h load and execute
  01h load but do not execute
  03h load overlay
  04h load and execute in background (European MS-DOS 4.0 only) "Exec & Go" (see also Function 80h in Chapter 25)
DS:DX -> ASCIZ program name (must include extension)
ES:BX -> parameter block (see Tables 24-42 to 24-44)
CX = mode (subfunction 04h only)
  0000h child placed in zombie mode after termination
  0001h child's return code discarded on termination

**Return Registers:**
CF clear if successful
  BX,DX destroyed
  if subfunction 01h, process ID set to new program's PSP; get with INT 21h Function 62h
CF set on error
  AX = error code (01h,02h,05h,08h,0Ah,0Bh) (see Table 24-2)

**Details:** DOS 2.x destroys all registers, including SS:SP.

Under ROM-based DOS, if no disk path characters (colons or slashes) are included in the program name, the name is searched for in the ROM module headers (see Table 24-46) before searching on disk.

For functions 00h and 01h, the calling process must ensure that there is enough unallocated memory available; if necessary, by releasing memory with Functions 49h or 4Ah.

For function 01h (which was undocumented prior to the release of MS-DOS 5.0), the AX value to be passed to the child program is put on top of the child's stack.

For function 03h, DOS assumes that the overlay is being loaded into memory allocated by the caller.

Some versions (such as DR-DOS 6.0) check the parameters and parameter block and return an error if an invalid value (such as an offset of FFFFh) is found.

Background programs under European MS-DOS 4.0 must use the new executable format.

New executables begin running with the following register values:

AX = environment segment
BX = offset of command tail in environment segment
CX = size of automatic data segment (0000h = 64K)
ES,BP = 0000h
DS = automatic data segment
SS:SP = initial stack

The command tail corresponds to an old executable's PSP:0081h and following, except that the 0Dh is turned into a NUL (00h); new format executables have no PSP.

Under the FlashTek X-32 DOS extender, only function 00h is supported and the pointers are passed in DS:EDX and ES:EBX.

DR-DOS 6 always loads .EXE-format programs with no fixups above the 64K mark to avoid the EXEPACK bug.

**BUGS:** DOS 2.00 assumes that DS points at the current program's PSP.

Load Overlay (subfunction 03h) loads up to 512 bytes too many if the file contains additional data after the actual overlay.

**Conflicts:** Viruses (Chapter 59), ELRES (Chapter 66)

**See also:** Functions 4B05h, 4Ch, and 4Dh; INT 2Eh (below); European MS-DOS Function 8Ah (Chapter 25); OS/2 Function 64h (Chapter 26)

**24-41 Names for the various executable types:**

| | |
|---|---|
| MZ | old-style DOS executable |
| NE | Windows or OS/2 1.x segmented ("new") executable |
| LE | Windows virtual device driver (VxD) linear executable |
| LX | variant of LE used in OS/2 2.x |
| W3 | Windows WIN386.EXE file; a collection of LE files |
| PE | Win32 (Windows NT and Win32s) portable executable based on Unix COFF |

**24-42 Format of EXEC parameter block for Subfunctions 00h,01h,04h:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of environment to copy for child process (copy caller's environment if 0000h) |
| 02h | DWORD | pointer to command tail to be copied into child's PSP |
| 06h | DWORD | pointer to first FCB to be copied into child's PSP |
| 0Ah | DWORD | pointer to second FCB to be copied into child's PSP |
| 0Eh | DWORD | (AL=01h) will hold subprogram's initial SS:SP on return |
| 12h | DWORD | (AL=01h) will hold entry point (CS:IP) on return |

### 24-43 Format of EXEC parameter block for AL=03h:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment at which to load overlay |
| 02h | WORD | relocation factor to apply to overlay if in .EXE format |

### 24-44 Format of EXEC parameter block for FlashTek X-32:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | PWORD | 48-bit far pointer to environment string |
| 06h | PWORD | 48-bit far pointer to command tail string |

### 24-45 Format of .EXE file header:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | .EXE signature, either “MZ” or “ZM” (5A4Dh or 4D5Ah) |
| 02h | WORD | number of bytes in last 512-byte page of executable |
| 04h | WORD | total number of 512-byte pages in executable (includes any partial last page) |
| 06h | WORD | number of relocation entries |
| 08h | WORD | header size in paragraphs |
| 0Ah | WORD | minimum paragraphs of memory to allocation in addition to executable’s size |
| 0Ch | WORD | maximum paragraphs to allocate in addition to executable’s size |
| 0Eh | WORD | initial SS relative to start of executable |
| 10h | WORD | initial SP |
| 12h | WORD | checksum (one’s complement of sum of all words in executable) |
| 14h | DWORD | initial CS:IP relative to start of executable |
| 18h | WORD | offset within header of relocation table<br>40h or greater for new-format (NE,LE,LX,W3,PE,etc.) executable |
| 1Ah | WORD | overlay number (normally 0000h = main program) |
| **---new executable---** | | |
| 1Ch | 4 BYTEs | *unknown* |
| 20h | WORD | behavior bits |
| 22h | 26 BYTEs | reserved for additional behavior info |
| 3Ch | DWORD | offset of new executable (NE,LE,etc) header within disk file, or 00000000h if plain MZ executable |
| **---Borland TLINK---** | | |
| 1Ch | 2 BYTEs | *unknown* (apparently always 01h 00h) |
| 1Eh | BYTE | signature FBh |
| 1Fh | BYTE | TLINK version (major in high nybble, minor in low nybble) |
| 20h | 2 BYTEs | *unknown* (v2.0 apparently always 72h 6Ah, v3.0+ seems always 6Ah 72h) |
| **---ARJ self-extracting archive---** | | |
| 1Ch | 4 BYTEs | signature “RJSX” (older versions, new signature is “aRJsfX” in the first 1000 bytes of the file) |
| **---LZEXE 0.90 compressed executable---** | | |
| 1Ch | 4 BYTEs | signature “LZ09” |
| **---LZEXE 0.91 compressed executable---** | | |
| 1Ch | 4 BYTEs | signature “LZ91” |
| **---PKLITE compressed executable---** | | |
| 1Ch | BYTE | minor version number |
| 1Dh | BYTE | bits 0-3: major version<br>bit 4: extra compression<br>bit 5: huge (multi-segment) file |
| 1Eh | 6 BYTEs | signature “PKLITE” (followed by copyright message) |

| | | |
|---|---|---|
| **---LHarc 1.x self-extracting archive---** | | |
| 1Ch | 4 BYTEs | *unused* |
| 20h | 3 BYTEs | jump to start of extraction code |
| 23h | 2 BYTEs | *unknown* |
| 25h | 12 BYTEs | signature “LHarc's SFX “ |
| **---LHA 2.x self-extracting archive---** | | |
| 1Ch | 8 BYTEs | *unknown* |
| 24h | 10 BYTEs | signature “LHa's SFX “ (v2.10) or “LHA's SFX “ (v2.13) |
| **---TopSpeed C 3.0 CRUNCH compressed file---** | | |
| 1Ch | DWORD | 018A0001h |
| 20h | WORD | 1565h |
| **---PKARC 3.5 self-extracting archive---** | | |
| 1Ch | DWORD | 00020001h |
| 20h | WORD | 0700h |
| **---BSA (Soviet archiver) self-extracting archive---** | | |
| 1Ch | WORD | 000Fh |
| 1Eh | BYTE | A7h |
| **---LARC self-extracting archive---** | | |
| 1Ch | 4 BYTEs | *unknown* |
| 20h | 11 BYTEs | “SFX by LARC “ |
| **---LH self-extracting archive---** | | |
| 1Ch | 8 BYTEs | *unknown* |
| 24h | 8 BYTEs | “LH's SFX “ |
| **---other linkers---** | | |
| 1Ch | varies | optional information |
| --- | | |
| N | N DWORDs | relocation items |

**Notes:** If the WORD at offset 02h is 4, it should be treated as 00h, since pre-1.10 versions of the MS linker set it that way.

If both minimum and maximum allocation (offset 0Ah/0Ch) are zero, the program is loaded as high in memory as possible.

The maximum allocation is set to FFFFh by default.

### 24-46 Format of ROM Module Header:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | ROM signature 55h, AAh |
| 02h | BYTE | size of ROM in 512-byte blocks |
| 03h | 3 BYTEs | POST initialization entry point (near JMP instruction) |
| 06h | ROM | Program Name List [array] |

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | length of ROM program's name (00h if end of name list) |
| 01h | N BYTEs | program name |
| N+1 | 3 BYTEs | program entry point (near JMP instruction) |

### 24-47 Format of new executable header:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | “NE” (4Eh 45h) signature |
| 02h | 2 BYTEs | linker version (major, then minor) |
| 04h | WORD | offset from start of this header to entry table |
| 06h | WORD | length of entry table in bytes |
| 08h | DWORD | file load CRC (0 in Borland's TPW) |

| | | |
|---|---|---|
| 0Ch | BYTE | program flags<br>bits 0-1 DGROUP type<br>0 none, 1 single shared, 2 multiple (unshared), 3 (null)<br>bit 2: global initialization<br>bit 3: protected mode only<br>bit 4 : 8086 instructions<br>bit 5 : 80286 instructions<br>bit 6 : 80386 instructions<br>bit 7 : 80x87 instructions |
| 0Dh | BYTE | application flags<br>bits 0-2: application type<br>001 full screen (not aware of Windows/P.M. API)<br>010 compatible with Windows/P.M. API<br>011 uses Windows/P.M. API<br>bit 3: is a Family Application (OS/2)<br>bit 5: 0=executable, 1=errors in image<br>bit 6: non-conforming program (valid stack is not maintained)<br>bit 7: DLL or driver rather than application (SS:SP info invalid, CS:IP points at FAR init routine called with AX=module handle which returns AX=0000h on failure, AX nonzero on successful initialization) |
| 0Eh | WORD | auto data segment index |
| 10h | WORD | initial local heap size |
| 12h | WORD | initial stack size (added to data seg, 0000h if SS != DS) |
| 14h | DWORD | program entry point (CS:IP), "CS" is index into segment table |
| 18h | DWORD | initial stack pointer (SS:SP), "SS" is segment index<br>if SS=automatic data segment and SP=0000h, the stack pointer is set to the top of the automatic data segment, just below the local heap |
| 1Ch | WORD | segment count |
| 1Eh | WORD | module reference count |
| 20h | WORD | length of nonresident names table in bytes |
| 22h | WORD | offset from start of this header to segment table |
| 24h | WORD | offset from start of this header to resource table |
| 26h | WORD | offset from start of this header to resident names table |
| 28h | WORD | offset from start of this header to module reference table |
| 2Ah | WORD | offset from start of this header to imported names table<br>(array of counted strings, terminated with a string of length 00h) |
| 2Ch | DWORD | offset from start of file to nonresident names table |
| 30h | WORD | count of moveable entry point listed in entry table |
| 32h | WORD | file alignment size shift count<br>0 is equivalent to 9 (default 512-byte pages) |
| 34h | WORD | number of resource table entries |
| 36h | BYTE | target operating system<br>00h unknown<br>01h OS/2<br>02h Windows<br>03h European MS-DOS 4.x<br>04h Windows 386<br>05h BOSS (Borland Operating System Services) |
| 37h | BYTE | other EXE flags |

| | | |
|---|---|---|
| | | bit 0: supports long filenames |
| | | bit 1: 2.X protected mode |
| | | bit 2: 2.X proportional font |
| | | bit 3: gangload area |
| 38h | WORD | offset to return thunks or start of gangload area |
| 3Ah | WORD | offset to segment reference thunks or length of gangload area |
| 3Ch | WORD | minimum code swap area size |
| 3Eh | 2 BYTEs | expected Windows version (minor version first) |

**Note:** This header and the additional tables pointed at by the header are documented in detail in the Windows 3.1 SDK Programmer's Reference, Volume 4.

### 24-48 Format of Codeview trailer (at end of executable):

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | signature 4E42h ('NB') |
| 02h | WORD | Microsoft debug info version number |
| 04h | DWORD | Codeview header offset |

### 24-49 Linear Executable CPU types:

| | |
|---|---|
| 01h | Intel 80286 or upwardly compatible |
| 02h | Intel 80386 or upwardly compatible |
| 03h | Intel 80486 or upwardly compatible |
| 04h | Intel Pentium (80586) or upwardly compatible |
| 20h | Intel i860 (N10) or compatible |
| 21h | Intel "N11" or compatible |
| 40h | MIPS Mark I (R2000, R3000) or compatible |
| 41h | MIPS Mark II (R6000) or compatible |
| 42h | MIPS Mark III (R4000) or compatible |

### 24-50 Format of Linear Executable (enhanced mode executable) header:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | "LE" (4Ch 45h) signature (Windows) |
| | | "LX" (4Ch 58h) signature (OS/2) |
| 02h | BYTE | byte order (00h = little-endian, nonzero = big-endian) |
| 03h | BYTE | word order (00h = little-endian, nonzero = big-endian) |
| 04h | DWORD | executable format level |
| 08h | WORD | CPU type (see Table 24-49) (see also INT 15h Function C9h in Chapter 4) |
| 0Ah | WORD | target operating system |
| | | 01h OS/2 |
| | | 02h Windows |
| | | 03h European DOS 4.0 |
| | | 04h Windows 386 |
| 0Ch | DWORD | module version |
| 10h | DWORD | module type (see Table 24-51) |
| 14h | DWORD | number of memory pages |
| 18h | Initial CS:EIP | |
| | DWORD | object number |
| | DWORD | offset |
| 20h | Initial SS:ESP | |
| | DWORD | object number |
| | DWORD | offset |
| 28h | DWORD | memory page size |
| 2Ch | DWORD | (Windows LE) bytes on last page |
| | | (OS/2 LX) page offset shift count |

| | | |
|---|---|---|
| 30h | DWORD | fixup section size |
| 34h | DWORD | fixup section checksum |
| 38h | DWORD | loader section size |
| 3Ch | DWORD | loader section checksum |
| 40h | DWORD | offset of object table (see Table 24-52) |
| 44h | DWORD | object table entries |
| 48h | DWORD | object page map table offset |
| 4CH | DWORD | object iterate data map offset |
| 50h | DWORD | resource table offset |
| 54h | DWORD | resource table entries |
| 58h | DWORD | resident names table offset |
| 5Ch | DWORD | entry table offset |
| 60h | DWORD | module directives table offset |
| 64h | DWORD | Module Directives entries |
| 68h | DWORD | Fixup page table offset |
| 6Ch | DWORD | Fixup record table offset |
| 70h | DWORD | imported modules name table offset |
| 74h | DWORD | imported modules count |
| 78h | DWORD | imported procedures name table offset |
| 7Ch | DWORD | per-page checksum table offset |
| 80h | DWORD | data pages offset |
| 84h | DWORD | preload page count |
| 88h | DWORD | non-resident names table offset |
| 8Ch | DWORD | non-resident names table length |
| 90h | DWORD | non-resident names checksum |
| 94h | DWORD | automatic data object |
| 98h | DWORD | debug information offset |
| 9Ch | DWORD | debug information length |
| A0h | DWORD | preload instance pages number |
| A4h | DWORD | demand instance pages number |
| A8h | DWORD | extra heap allocation |
| ACh | 20 BYTEs | reserved |
| C0h | WORD | device ID (MS-Windows VxD only) |
| C2h | WORD | DDK version (MS-Windows VxD only) |

**Note:** This format is used by EMM386.EXE, QEMM, and Windows 3.0 Enhanced Mode drivers.

**24-51 Bitfields for Linear Executable module type:**

bit 2: initialization (only for DLLs)
    0 = global
    1 = per-process
bit 4: no internal fixups in executable image
bit 5: no external fixups in executable image
bits 8,9,10:
    0 = unknown
    1 = incompatible with PM windowing
    2 = compatible with PM windowing
    3 = uses PM windowing API
    } (only for programs)
bit 13: module not loadable (only for programs)
bits 17,16,15: module type
    000 program

## INT 21h Function 4Bh

001 library (DLL)
011 protected memory library module
100 physical device driver
110 virtual device driver

bit 30: per-process library termination (requires valid CS:EIP, can't be set for .EXE)

**24-52 Format of linear executable object table entry:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | virtual size in bytes |
| 04h | DWORD | relocation base address |
| 08h | DWORD | object flags (see Table 24-53) |
| 0Ch | DWORD | page map index |
| 10h | DWORD | page map entries |
| 14h | 4 BYTEs | *reserved* (apparently always zeros) |

**24-53 Bitfields for object flags:**

| | |
|---|---|
| bit 0 | readable |
| bit 1 | writable |
| bit 2 | executable |
| bit 3 | resource |
| bit 4 | discardable |
| bit 5 | shared |
| bit 6 | preloaded |
| bit 7 | invalid |
| bit 8 | -9 type<br>00 normal<br>01 zero-filled<br>10 resident<br>11 resident and contiguous |
| bit 10 | resident and long-lockable |
| bit 11 | reserved |
| bit 12 | 16:16 alias required |
| bit 13 | "BIG" (Huge: 32-bit) |
| bit 14 | conforming |
| bit 15 | "OBJECT_I/O_PRIVILEGE_LEVEL" |
| bits 16-31 | reserved |

**24-54 Format of object page map table entry:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | *unknown* (usually 00h) |
| 01h | WORD | (big-endian) index to fixup table<br>0000h if no relocation info |
| 03h | BYTE | type (00h hard copy in file, 03h some relocation needed) |

**24-55 Format of resident names table entry:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | length of name |
| 01h | N BYTEs | name |
| N+1 | 3 BYTEs | *unknown* |

**24-56 Format of LE linear executable entry table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | number of entries in table |
| 01h | 10N BYTEs | entry records (see Table 24-57) |

**24-57 Format of LE linear executable entry record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | bit flags |
| | | bit 0: non-empty bundle |
| | | bit 1: 32-bit entry |
| 01h | WORD | object number |
| 03h | BYTE | entry type flags |
| | | bit 0: exported |
| | | bit 1: uses single data rather than instance |
| | | bit 2: reserved |
| | | bits 3-7: number of stack parameters |
| 04h | DWORD | offset of entry point |
| 08h | 2 BYTEs | *unknown* |

**Note:** Empty bundles (bit flags at 00h = 00h) are used to skip unused indices, and do not contain the remaining nine bytes.

**24-58 Format of LX linear executable entry table [array]:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | number of bundles following (00h = end of entry table) |
| 01h | BYTE | bundle type |
| | | 00h empty |
| | | 01h 16-bit entry |
| | | 02h 286 callgate entry |
| | | 03h 32-bit entry |
| | | 04h forwarder entry |
| | | bit 7 set if additional parameter typing information is present |

---**bundle type 00h**---

no additional fields

---**bundle type 01h**---

| | | |
|---|---|---|
| 02h | WORD | object number |
| 04h | BYTE | entry flags |
| | | bit 0: exported |
| | | bits 7-3: number of stack parameters |
| 05h | WORD | offset of entry point in object (shifted by page size shift) |

---**bundle type 02h**---

| | | |
|---|---|---|
| 02h | WORD | object number |
| 04h | BYTE | entry flags |
| | | bit 0: exported |
| | | bits 7-3: number of stack parameters |
| 05h | WORD | offset of entry point in object |
| 07h | WORD | reserved for callgate selector (used by loader) |

---**bundle type 03h**---

| | | |
|---|---|---|
| 02h | WORD | object number |
| 04h | BYTE | entry flags |
| | | bit 0: exported |
| | | bits 7-3: number of stack parameters |
| 05h | DWORD | offset of entry point in object |

---**bundle type 04h**---

| | | |
|---|---|---|
| 02h | WORD | reserved |
| 04h | BYTE | forwarder flags |

| | | |
|---|---|---|
| | | bit 0: import by ordinal<br>bits 7-1 reserved |
| 05h | WORD | module ordinal<br>(forwarder's index into Import Module Name table) |
| 07h | DWORD | procedure name offset or import ordinal number |

**Note:** All fields after the first two bytes are repeated N times.

### 24-59 Bitfields for linear executable fixup type:

| | |
|---|---|
| bit 7 | ordinal is BYTE rather than WORD |
| bit 6 | 16-bit rather than 8-bit object number/module ordinal |
| bit 5 | addition with DWORD rather than WORD |
| bit 4 | relocation info has size with new two bytes at end |
| bit 3 | reserved (0) |
| bit 2 | set if add to destination, clear to replace destination |
| bits 1-0 | type<br>00 internal fixup<br>01 external fixup, imported by ordinal<br>10 external fixup, imported by name<br>11 internal fixup via entry table |

### 24-60 Format of linear executable fixup record:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | type<br>bits 7-4: modifier (0001 single, 0011 multiple)<br>bits 3-0: type<br>0000 byte offset<br>0010 word segment<br>0011 16-bit far pointer (DWORD)<br>0101 16-bit offset<br>0110 32-bit far pointer (PWORD)<br>0111 32-bit offset<br>1000 near call or jump, WORD/DWORD based on seg attrib |
| 01h | BYTE | linear executable fixup type (see Table 24-59) |
| **---if single type---** | | |
| 02h | WORD | offset within page |
| 04h | relocation information | |
| | ---internal fixup--- | |
| | BYTE | object number |
| | ---external,ordinal--- | |
| | BYTE | one-based module number in Import Module table |
| | BYTE/WORD | ordinal number |
| | WORD/DWORD | value to add (only present if modifier bit 4 set) |
| | ---external,name--- | |
| | BYTE | one-based module number in Import Module table |
| | WORD | offset in Import Procedure names |
| | WORD/DWORD | value to add (only present if modifier bit 4 set) |
| **---if multiple type---** | | |
| 02h | BYTE | number of items |
| 03h | varies | relocation info as for “single” type (see above) |
| | N WORDs | offsets of items to relocate |

**24-61 Format of old Phar Lap .EXP file header:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | "MP" (4Dh 50h) signature |
| 02h | WORD | remainder of image size / page size (page size = 512h) |
| 04h | WORD | size of image in pages |
| 06h | WORD | number of relocation items |
| 08h | WORD | header size in paragraphs |
| 0Ah | WORD | minimum number of extra 4K pages to be allocated at the end of program, when it is loaded |
| 0Ch | WORD | maximum number of extra 4K pages to be allocated at the end of program, when it is loaded |
| 0Eh | DWORD | initial ESP |
| 12h | WORD | word checksum of file |
| 14h | DWORD | initial EIP |
| 18h | WORD | offset of first relocation item |
| 1Ah | WORD | overlay number |
| 1Ch | WORD | *unknown* (wants to be 1) |

**24-62 Format of new Phar Lap .EXP file header:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | signature ("P2" for 286 .EXP executable, "P3" for 386 .EXP) |
| 02h | WORD | level (01h flat-model file, 02h multisegmented file) |
| 04h | WORD | header size |
| 06h | DWORD | file size in bytes |
| 0Ah | WORD | checksum |
| 0Ch | DWORD | offset of run-time parameters within file |
| 10h | DWORD | size of run-time parameters in bytes |
| 14h | DWORD | offset of relocation table within file |
| 18h | DWORD | size of relocation table in bytes |
| 1Ch | DWORD | offset of segment information table within file |
| 20h | DWORD | size of segment information table in bytes |
| 24h | WORD | size of segment information table entry in bytes |
| 26h | DWORD | offset of load image within file |
| 2Ah | DWORD | size of load image on disk |
| 2Eh | DWORD | offset of symbol table within file |
| 32h | DWORD | size of symbol table in bytes |
| 36h | DWORD | offset of GDT within load image |
| 3Ah | DWORD | size of GDT in bytes |
| 3Eh | DWORD | offset of LDT within load image |
| 42h | DWORD | size of LDT in bytes |
| 46h | DWORD | offset of IDT within load image |
| 4Ah | DWORD | size of IDT in bytes |
| 4Eh | DWORD | offset of TSS within load image |
| 52h | DWORD | size of TSS in bytes |
| 56h | DWORD | minimum number of extra bytes to be allocated at end of program (level 1 executables only) |
| 5Ah | DWORD | maximum number of extra bytes to be allocated at end of program (level 1 executables only) |
| 5Eh | DWORD | base load offset (level 1 executables only) |
| 62h | DWORD | initial ESP |

| | | |
|---|---|---|
| 66h | WORD | initial SS |
| 68h | DWORD | initial EIP |
| 6Ch | WORD | initial CS |
| 6Eh | WORD | initial LDT |
| 70h | WORD | initial TSS |
| 72h | WORD | flags |
| | | bit 0: load image is packed |
| | | bit 1: 32-bit checksum is present |
| | | bits 4-2: type of relocation table |
| 74h | DWORD | memory requirements for load image |
| 78h | DWORD | 32-bit checksum (optional) |
| 7Ch | DWORD | size of stack segment in bytes |
| 80h | 256 BYTEs | reserved (0) |

**24-63 Format of Phar Lap segment information table entry:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | selector number |
| 02h | WORD | flags |
| 04h | DWORD | base offset of selector |
| 08h | DWORD | minimum number of extra bytes to be allocated to the segment |

**24-64 Format of 386|DOS-Extender run-time parameters:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | signature "DX" (44h 58h) |
| 02h | WORD | minimum number of real-mode parameters to leave free at run time |
| 04h | WORD | maximum number of real-mode parameters to leave free at run time |
| 06h | WORD | minimum interrupt buffer size in KB |
| 08h | WORD | maximum interrupt buffer size in KB |
| 0Ah | WORD | number of interrupt stacks |
| 0Ch | WORD | size in KB of each interrupt stack |
| 0Eh | DWORD | offset of byte past end of real-mode code and data |
| 12h | WORD | size in KB of call buffers |
| 14h | WORD | flags |
| | | bit 0: file is virtual memory manager |
| | | bit 1: file is a debugger |
| 16h | WORD | unprivileged flag (if nonzero, executes at ring 1, 2, or 3) |
| 18h | 104 BYTEs | reserved (0) |

**24-65 Format of Phar Lap repeat block header:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | byte count |
| 02h | BYTE | repeat string length |

---

## INT 21h Function 4B05h

*DOS 5+*

### *SET EXECUTION STATE*

**Registers at call:**
AX = 4B05h
DS:DX -> execution state structure
(see Table 24-66)

**Return Registers:**
CF clear if successful
AX = 0000h
CF set on error
AX = error code (see Table 24-2)

**Details:** This function is used by programs which intercept Function 4B00h to prepare new programs for execution (including setting the DOS version number). No DOS, BIOS or other soft-

ware interrupt may be called after return from this call before commencement of the child process. If DOS is running in the HMA, A20 is turned off on return from this call.
**See also:** Function 4Bh

**24-66 Format of execution state structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | reserved (00h) |
| 02h | WORD | type flags<br>bit 0: program is an .EXE<br>bit 1: program is an overlay |
| 04h | DWORD | pointer to ASCIZ name of program file |
| 08h | WORD | PSP segment of new program |
| 0Ah | DWORD | starting CS:IP of new program |
| 0Eh | DWORD | program size including PSP |

### INT 21h Function 4Ch — *DOS 2+*
### *"EXIT" - TERMINATE WITH RETURN CODE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 4Ch | never returns |
| AL = return code | |

**Details:** Unless the process is its own parent (see Function 26h, offset 16h in PSP), all open files are closed and all memory belonging to the process is freed.

All network file locks should be removed before calling this function.

**Conflicts:** Headroom (Chapter 49)
**See also:** Functions 00h, 26h, 4Bh, and 4Dh; INT 20h; INT 22h; TopView INT 15h Function 12h Subfunction 02h (Chapter 44); HP 95LX INT 60h Function 0601h (Chapter 20)

### INT 21h Function 4Dh — *DOS 2+*
### *GET RETURN CODE (ERRORLEVEL)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 4Dh | AH = termination type (see Table 24-67) |
| | AL = return code |

**Details:** The word in which DOS stores the return code is cleared after being read by this function, so the return code can only be retrieved once.

COMMAND.COM stores the return code of the last external command it executed as ERRORLEVEL.

This call should not be used if the child was started with Function 4B04h; use Function 8Ah (Chapter 25) instead.

The following sequence will close a Virtual DOS Machine under OS/2 2.0:

```
MOV AH,4Dh
INT 21h
HLT
DB 02h,0FDh
```

This sequence is the only way to close a specific VDM which was booted from floppy or a disk image.

**See also:** Functions 4Bh and 4Ch, European MS-DOS Function 8Ah (Chapter 25)

**24-67 Termination types:**

| | |
|---|---|
| 00h | normal (INT 20h, INT 21h Function 00h, or INT 21h Function 4Ch) |
| 01h | control-C abort |
| 02h | critical error abort |
| 03h | terminate and stay resident (INT 21h Function 31h or INT 27h) |

## INT 21h Function 4Eh *DOS 2+*
### *"FINDFIRST" - FIND FIRST MATCHING FILE*

**Registers at call:**
AH = 4Eh
AL = special flag for use by APPEND (see note below)
CX = file attribute mask (see Table 24-21), bits 0 and 5 ignored
 0088h (Novell DOS 7) find first deleted file
DS:DX -> ASCIZ file specification (may include path and wildcards)

**Return Registers:**
CF clear if successful
 [DTA] = FindFirst data block (see Table 24-68)
CF set on error
 AX = error code (02h,03h,12h) (see Table 24-2)

**Details:** For search attributes other than 08h, all files with at *most* the specified combination of hidden, system, and directory attributes will be returned. Under DOS 2.x, searching for attribute 08h (volume label) will also return normal files, while under DOS 3+ only the volume label (if any) will be returned.

This call also returns successfully if given the name of a character device without wildcards. DOS 2.x returns attribute 00h, size 0, and the current date and time. DOS 3+ returns attribute 40h and the current date and time.

Immediately after an INT 2Fh Function B711h (APPEND return found name, see Chapter 27), the name at DS:DX will be overwritten; if AL=00h on entry, the actual found pathname will be stored, otherwise, the actual found path will be prepended to the original filespec without a path.

Under LANtastic, this call may be used to obtain a list of a server's shared resources by searching for "\\SERVER*.*"; a list of printer resources may be obtained by searching for "\\SERVER\@*.*".

Under the FlashTek X-32 DOS extender, the filespec pointer is in DS:EDX

**BUGS:** Under DOS 3.x and 4.x, the second and subsequent calls to this function with a character device name (no wildcards) and search attributes which include the volume-label bit (08h) will fail unless there is an intervening DOS call which implicitly or explicitly performs a directory search without the volume-label bit. Such implicit searches are performed by CREATE (Function 3Ch), OPEN (Function 3Dh), UNLINK (Function 41h), and RENAME (Function 56h).

DR-DOS 3.41 and 5.0 return the Directory attribute for the volume label.

**Conflicts:** WILDUNIX (Chapter 66)

**See also:** Functions 11h, 4Fh, and 4301h; APPEND INT 2Fh Function B711h (Chapter 27); Network Redirector INT 2Fh Function 111Bh (NI)

**24-68 Format of FindFirst data block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| **---PC-DOS 3.10, PC-DOS 4.01, MS-DOS 3.2/3.3/5.0---** | | |
| 00h | BYTE | drive letter (bits 0-6), remote if bit 7 set |
| 01h | 11 BYTEs | search template |
| 0Ch | BYTE | search attributes |
| **---DOS 2.x** (*and some DOS 3.x*)--- | | |
| 00h | BYTE | search attributes |
| 01h | BYTE | drive letter |
| 02h | 11 BYTEs | search template |
| **---WILDUNIX.COM---** | | |
| 00h | 12 BYTEs | 15-character wildcard search pattern and drive letter (packed) |
| 0Ch | BYTE | search attributes |
| **---DOS 2.x and most 3.x---** | | |

| | | |
|---|---|---|
| 0Dh | WORD | entry count within directory |
| 0Fh | DWORD | *pointer to DTA* |
| 13h | WORD | cluster number of start of parent directory |
| **---PC-DOS 4.01, MS-DOS 3.2/3.3/5.0---** | | |
| 0Dh | WORD | entry count within directory |
| 0Fh | WORD | cluster number of start of parent directory |
| 11h | 4 BYTEs | reserved |
| **---all versions, documented fields---** | | |
| 15h | BYTE | attribute of file found |
| 16h | WORD | file time<br>bits 11-15: hour<br>bits 5-10: minute<br>bits 0-4: seconds/2 |
| 18h | WORD | file date<br>bits 9-15: year-1980<br>bits 5-8: month<br>bits 0-4: day |
| 1Ah | DWORD | file size |
| 1Eh | 13 BYTEs | ASCIZ filename+extension |

## INT 21h Function 4Fh

*DOS 2+*

### *"FINDNEXT" - FIND NEXT MATCHING FILE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 4Fh | CF clear if successful |
| [DTA] = data block from previous FindFirst or FindNext call | CF set on error<br>AX = error code (12h) (see Table 24-2) |

**Details:** Under Novell DOS 7, if the FindFirst call (Function 4Eh) had CX=0088h, then the next matching deleted file will be returned.

**BUG:** DR-DOS 3.41 and 5.0 return the Directory attribute for the volume label.

**See also:** Functions 12h and 4Eh

## INT 21h Function 50h

*DOS 2+ internal*

### *SET CURRENT PROCESS ID (SET PSP ADDRESS)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 50h | none |
| BX = segment of PSP for new process | |

**Details:** DOS uses the current PSP address to determine which processes own files and memory; it corresponds to process identifiers used by other OSs.

Under DOS 2.x, this function cannot be invoked inside an INT 28h handler without setting the Critical Error flag. Under MS-DOS 3+ and DR-DOS 3.41+, this function does not use any of the DOS-internal stacks and may thus be called at any time, even during another INT 21h call.

Some Microsoft applications such as Quick C 2.51 use segments of 0000h and FFFFh and direct access to the SDA (see Tables 24-108 and 24-110) to test whether they are running under MS-DOS rather than a compatible OS; although one should only call this function with valid PSP addresses, any program hooking it should be prepared to handle invalid addresses.

This function is supported by the OS/2 compatibility box.

This call was undocumented prior to the release of DOS 5.0.

**See also:** Functions 26h, 51h, and 62h

### INT 21h Function 51h *DOS 2+ internal*
### *GET CURRENT PROCESS ID (GET PSP ADDRESS)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 51h | BX = segment of PSP for current process |

**Details:** DOS uses the current PSP address to determine which processes own files and memory; it corresponds to process identifiers used by other OSs.

Under DOS 2.x, this function cannot be invoked inside an INT 28h handler without setting the Critical Error flag. Under DOS 3+, this function does not use any of the DOS-internal stacks and may thus be called at any time, even during another INT 21h call.

This function is supported by the OS/2 compatibility box.

This call was undocumented prior to the release of DOS 5.0; it is identical to the documented Function 62h.

**See also:** Functions 26h, 50h, and 62h

### INT 21h Function 52h *DOS 2+ internal*
### *"SYSVARS" - GET LIST OF LISTS* *(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 52h | ES:BX -> DOS list of lists (see Table 24-69) |

**Details:** On return, ES points at the DOS data segment (see also INT 2Fh Function 1203h).

This function is partially supported by the OS/2 v1.1 compatibility box (however, most pointers are FFFFh:FFFFh, LASTDRIVE is FFh, and the NUL header "next" pointer is FFFFh:FFFFh).

**Conflicts:** Leapfrog virus (Chapter 59)

**See also:** INT 2Fh Function 1203h

**24-69 Format of List of Lists:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -22 | WORD | (DOS v?+) LRU counter for FCB caching |
| -20 | WORD | (DOS v?+) LRU counter for FCB opens |
| -18 | DWORD | (DOS v?+) address of OEM function handler (see INT 21h Function F8h) FFFFh:FFFFh if not installed or not available |
| -14 | WORD | (DOS v?+) offset in DOS CS of code to return from INT 21h call |
| -12 | WORD | (DOS 3.1+) sharing retry count (see Function 440Bh) |
| -10 | WORD | (DOS 3.1+) sharing retry delay (see Function 440Bh) |
| -8 | DWORD | (DOS 3+) pointer to current disk buffer |
| -4 | WORD | (DOS 3+) pointer in DOS code segment of unread CON input when CON is read via a handle, DOS reads an entire line, and returns the requested portion, buffering the rest for the next read. 0000h indicates no unread input |
| -2 | WORD | segment of first memory control block |
| 00h | DWORD | pointer to first Drive Parameter Block (see Table 24-15) |
| 04h | DWORD | pointer to first System File Table (see Tables 24-79 and 24-80) |
| 08h | DWORD | pointer to active CLOCK$ device's header (most recently loaded driver with CLOCK bit set) |
| 0Ch | DWORD | pointer to active CON device's header (most recently loaded driver with STDIN bit set) |
| ---DOS 2.x--- | | |
| 10h | BYTE | number of logical drives in system |
| 11h | WORD | maximum bytes/block of any block device |
| 13h | DWORD | pointer to first disk buffer (see Table 24-86) |
| 17h | 18 BYTEs | actual NUL device driver header (not a pointer!) |

| | | |
|---|---|---|
| | | NUL is always the first device on DOS's linked list of device drivers.<br>(see Table 24-83) |

**---DOS 3.0---**

| | | |
|---|---|---|
| 10h | BYTE | number of block devices |
| 11h | WORD | maximum bytes/block of any block device |
| 13h | DWORD | pointer to first disk buffer (see Table 24-87) |
| 17h | DWORD | pointer to array of current directory structures (see Table 24-81) |
| 1Bh | BYTE | value of LASTDRIVE command in CONFIG.SYS (default 5) |
| 1Ch | DWORD | pointer to STRING= workspace area |
| 20h | WORD | size of STRING area (the x in STRING=x from CONFIG.SYS) |
| 22h | DWORD | pointer to FCB table |
| 26h | WORD | the y in FCBS=x,y from CONFIG.SYS |
| 28h | 18 BYTEs | actual NUL device driver header (not a pointer!)<br>NUL is always the first device on DOS's linked list of device drivers.<br>(see Table 24-83) |

**---DOS 3.1-3.3---**

| | | |
|---|---|---|
| 10h | WORD | maximum bytes per sector of any block device |
| 12h | DWORD | pointer to first disk buffer in buffer chain (see Table 24-87) |
| 16h | DWORD | pointer to array of current directory structures (see Table 24-81) |
| 1Ah | DWORD | pointer to system FCB tables (see Table 24-79) |
| 1Eh | WORD | number of protected FCBs (the y in the CONFIG.SYS FCBS=x,y) |
| 20h | BYTE | number of block devices installed |
| 21h | BYTE | number of available drive letters (largest of 5, installed block devices, and CONFIG.SYS LASTDRIVE=). Also size of current directory structure array. |
| 22h | 18 BYTEs | actual NUL device driver header (not a pointer!)<br>NUL is always the first device on DOS's linked list of device drivers.<br>(see Table 24-83) |
| 34h | BYTE | number of JOIN'ed drives |

**---DOS 4.x---**

| | | |
|---|---|---|
| 10h | WORD | maximum bytes per sector of any block device |
| 12h | DWORD | pointer to disk buffer info record (see Tables 24-88, 24-89, and 24-93) |
| 16h | DWORD | pointer to array of current directory structures (see Table 24-81) |
| 1Ah | DWORD | pointer to system FCB tables (see Table 24-80) |
| 1Eh | WORD | number of protected FCBs (the y in the CONFIG.SYS FCBS=x,y)<br>(always 00h for DOS 5.0) |
| 20h | BYTE | number of block devices installed |
| 21h | BYTE | number of available drive letters (largest of 5, installed block devices, and CONFIG.SYS LASTDRIVE=). Also size of current directory structure array. |
| 22h | 18 BYTEs | actual NUL device driver header (not a pointer!)<br>NUL is always the first device on DOS's linked list of device drivers.<br>(see Table 24-83) |
| 34h | BYTE | number of JOIN'ed drives |
| 35h | WORD | pointer within IBMDOS code segment to list of special program names (see Table 24-94)<br>(always 0000h for DOS 5.0) |
| 37h | DWORD | pointer to FAR routine for resident IFS utility functions<br>may be called by any IFS driver which does not wish to service functions 20h or 24h-28h itself |

| | | |
|---|---|---|
| 3Bh | DWORD | pointer to chain of IFS (installable file system) drivers |
| 3Fh | WORD | the x in BUFFERS x,y (rounded up to multiple of 30 if in EMS) |
| 41h | WORD | number of lookahead buffers (the y in BUFFERS x,y) |
| 43h | BYTE | boot drive (1=A:) |
| 44h | BYTE | flag: 01h to use DWORD moves (80386+), 00h otherwise |
| 45h | WORD | extended memory size in KB |
| **---DOS 5.0-6.0---** | | |
| 10h | 39 BYTEs | as for DOS 4.x (see above) |
| 37h | DWORD | pointer to SETVER program list or 0000h:0000h |
| 3Bh | WORD | (DOS=HIGH) offset in DOS CS of function to fix A20 control when executing special .COM format |
| 3Dh | WORD | PSP of most-recently EXECed program if DOS in HMA, 0000h if low |
| 3Fh | 8 BYTEs | as for DOS 4.x (see above) |

**24-70 Format of memory control block (see also below):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | block type: 5Ah if last block in chain, otherwise 4Dh |
| 01h | WORD | PSP segment of owner or special value (see Table 24-71) |
| 03h | WORD | size of memory block in paragraphs |
| 05h | 3 BYTEs | unused by MS-DOS<br>(386MAX) if locked-out block, region start/prev region end |
| **---DOS 2.x,3.x---** | | |
| 08h | 8 BYTEs | unused |
| **---DOS 4+ ---** | | |
| 08h | 8 BYTEs | ASCII program name if PSP memory block or DR-DOS UMB, else garbage<br>null-terminated if less than 8 characters |

**Notes:** Under DOS 3.1+, the first memory block is the DOS data segment, containing installable drivers, buffers, etc. Under DOS 4+ it is divided into subsegments, each with its own memory control block (see Table 24-73), the first of which is at offset 0000h.

For DOS 5+, blocks owned by DOS may have either "SC" or "SD" in bytes 08h and 09h. "SC" is system code or locked-out inter-UMB memory, "SD" is system data, device drivers, etc.

Some versions of DR-DOS use only seven characters of the program name, placing a NUL in the eighth byte.

**24-71 Special MCB owner values:**

| | |
|---|---|
| 0000h | free |
| 0006h | DR-DOS XMS UMB |
| 0007h | DR-DOS excluded upper memory ("hole") |
| 0008h | belongs to DOS |
| FFFAh | 386MAX UMB control block (see Table 40-11) |
| FFFDh | 386MAX locked-out memory |
| FFFEh | 386MAX UMB (immediately follows its control block) |
| FFFFh | 386MAX 6.01 device driver |

**24-72 Format of MS-DOS 5+ UMB control block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | type: 5Ah if last block in chain, 4Dh otherwise |
| 01h | WORD | first available paragraph in UMB if control block at start of UMB, 000Ah if control block at end of UMB |
| 03h | WORD | length in paragraphs of following UMB or locked-out region |
| 05h | 3 BYTEs | unused |
| 08h | 8 BYTEs | block type name: "UMB" if start block, "SM" if end block in UMB |

### 24-73 Format of DOS 4+ data segment subsegment control blocks:

| Offset | Size | Description |
|---|---|---|
| 00h | BYTE | subsegment type (blocks typically appear in this order)<br>"D" device driver<br>"E" device driver appendage<br>"I" IFS (Installable File System) driver<br>"F" FILES= control block storage area (for FILES>5)<br>"X" FCBS= control block storage area, if present<br>"C" BUFFERS EMS workspace area (if BUFFERS /X option used)<br>"B" BUFFERS= storage area<br>"L" LASTDRIVE= current directory structure array storage area<br>"S" STACKS= code and data area, if present (see Table 24-74)<br>"T" INSTALL= transient code |
| 01h | WORD | paragraph of subsegment start (usually the next paragraph) |
| 03h | WORD | size of subsegment in paragraphs |
| 05h | 3 BYTEs | unused |
| 08h | 8 BYTEs | for types "D" and "I", base name of file from which the driver was loaded (unused for other types) |

### 24-74 Format of data at start of STACKS code segment (if present):

| Offset | Size | Description |
|---|---|---|
| 00h | WORD | *unknown* |
| 02h | WORD | number of stacks (the x in STACKS=x,y) |
| 04h | WORD | size of stack control block array (should be 8*x) |
| 06h | WORD | size of each stack (the y in STACKS=x,y) |
| 08h | DWORD | pointer to STACKS data segment |
| 0Ch | WORD | offset in STACKS data segment of stack control block array |
| 0Eh | WORD | offset in STACKS data segment of last element of that array |
| 10h | WORD | offset in STACKS data segment of the entry in that array for the next stack to be allocated (initially same as value in 0Eh and works its way down in steps of 8 to the value in 0Ch as hardware interrupts pre-empt each other) |

**Note:** The STACKS code segment data may, if present, be located as follows:

DOS 3.2: The code segment data is at a paragraph boundary fairly early in the IBMBIO segment (seen at 0070:0190h)

DOS 3.3: The code segment is at a paragraph boundary in the DOS data segment, which may be determined by inspecting the segment pointers of the vectors for those of interrupts 02h, 08h-0Eh, 70h, 72-77h which have not been redirected by device drivers or TSRs.

DOS 4+ Identified by sub-segment control block type "S" within the DOS data segment.

**See also:** STACKMAN INT B4h

### 24-75 Format of array elements in STACKS data segment:

| Offset | Size | Description |
|---|---|---|
| 00h | BYTE | status: 00h=free, 01h=in use, 03h=corrupted by overflow of higher stack. |
| 01h | BYTE | not used |
| 02h | DWORD | previous SS:SP |
| 06h | WORD | pointer to word at top of stack (new value for SP). The word at the top of the stack is preset to point back to this control block. |

**24-76 DOS 3.1+ SHARE.EXE hooks:**

(offsets from first system file table--pointed at by ListOfLists+04h)

| *Offset* | *Size* | *Description* |
|---|---|---|
| -3Ch | DWORD | pointer to *unknown* FAR routine<br>**Note:** not called by MS-DOS 3.3, set to 0000h:0000h by SHARE 3.3+ |
| -38h | DWORD | pointer to FAR routine called on opening file<br>**Call:** internal DOS location (see Tables 24-108 and 24-110)<br>points at name of file just opened<br>**Returns:**<br>CF clear if successful<br>CF set on error<br>AX = DOS error code (24h) (see Table 24-2) |
| -34h | DWORD | pointer to FAR routine called on closing file<br>**Call:** ES:DI -> system file table entry for file |
| -30h | DWORD | pointer to FAR routine to close all files for given computer (called by Function 5D03h) |
| -2Ch | DWORD | pointer to FAR routine to close all files for given process (called by Function 5D04h) |
| -28h | DWORD | pointer to FAR routine to close file by name (called by Function 5D02h)<br>**Call:** DS:SI -> DOS parameter list (see Table 24-107)<br>DPL's DS:DX -> name of file to close.<br>**Returns:**<br>CF clear if successful<br>CF set on error<br>AX = DOS error code (03h) (see Table 24-2) |
| -24h | DWORD | pointer to FAR routine to lock region of file<br>**Call:** BX = file handle<br>CX:DX = starting offset<br>SI:AX = size<br>**Returns:**<br>CF set on error<br>AL = DOS error code (21h) (see Table 24-2)<br>**Details:** This function is not called if the file is marked as remote. |
| -20h | DWORD | pointer to FAR routine to unlock region of file<br>**Call:** BX = file handle<br>CX:DX = starting offset<br>SI:AX = size<br>**Returns:**<br>CF set on error<br>AL = DOS error code (21h) (see Table 24-2)<br>**Details:** This function is not called if the file is marked as remote. |
| -1Ch | DWORD | pointer to FAR routine to check if file region is locked<br>**Call:** ES:DI -> system file table entry for file<br>CX = length of region from current position in file<br>**Returns:**<br>CF set if any portion of region locked<br>AX = 0021h |
| -18h | DWORD | pointer to FAR routine to get open file list entry (called by Function 5D05h)<br>**Call:** DS:SI -> DOS parameter list (see Table 24-107) |

| | | |
|---|---|---|
| | | DPL's BX = index of sharing record<br>DPL's CX = index of SFT in SFT chain of sharing record<br>**Returns:**<br>CF set on error or not loaded<br>AX = DOS error code (12h) (see Table 24-2)<br>CF clear if successful<br>ES:DI -> filename<br>CX = number of locks owned by specified SFT<br>BX = network machine number<br>DX destroyed |
| -14h | DWORD | pointer to FAR routine for *updating FCB from SFT*<br>**Call:** DS:SI -> unopened FCB<br>ES:DI -> system file table entry<br>**Returns:**<br>BL = *C0h*<br>**Details:** This function copies the following fields from SFT to FCB: starting cluster of file (0Bh->1Ah), sharing record offset (33h->1Ch), and file attribute (04h->1Eh) |
| -10h | DWORD | pointer to FAR routine to *get first cluster of FCB file*<br>**Call:** ES:DI -> system file table entry<br>DS:SI -> FCB<br>**Returns:**<br>CF set if SFT closed or sharing record offsets mismatched<br>CF clear if successful<br>BX = starting cluster number from FCB |
| -0C h | DWORD | pointer to FAR routine to close file if duplicate for process<br>**Call:** DS:SI -> system file table<br>**Returns:**<br>AX = number of handle in JFT which already uses SFT<br>**Details:** This function is called during the open/create of a file. If the SFT was opened with inheritance enabled and sharing mode 111, this function does something to all other SFTs owned by the same process which have the same file open mode and sharing record |
| -08h | DWORD | pointer to FAR routine for closing file<br>**Details:** This function closes various handles referring to the file most-recently opened |
| -04h | DWORD | pointer to FAR routine to update directory info in related SFT entries<br>**Call:** ES:DI -> system file table entry for file (see Tables 24-79 and 24-80)<br>AX = subfunction (apply to each related SFT)<br>00h: update time stamp (offset 0Dh) and date stamp (offset 0Fh)<br>01h: update file size (offset 11h) and starting cluster (offset 0Bh). Sets last-accessed cluster fields to start of file if file never accessed<br>02h: as function 01h, but last-accessed fields always changed<br>03h: do both functions 00h and 02h<br>**Details:** This function follows the pointer at offset 2Bh in system file table entries. It is a NOP if the file was opened with no-inherit or via FCB. |

**Details:** Most of the above hooks (except -04h, -14h, -18h, and -3Ch) assume either that SS=DOS DS or SS=DS=DOS DS and directly access DOS-internal data

## INT 21h Function 52h

**24-77 Format of sharing record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | flag: 00h free block, 01h allocated block, FFh end marker |
| 01h | WORD | size of block |
| 03h | BYTE | checksum of pathname (including NUL)<br>if sum of ASCII values is N, checksum is (N/256 + N%256) |
| 04h | WORD | offset in SHARE's DS of first Record Lock Record (see Table 24-78) |
| 06h | DWORD | pointer to start of system file table chain for file |
| 0Ah | WORD | unique sequence number |
| 0Ch | varies | ASCIZ full pathname |

**24-78 Format of Record Lock Record (SHARE.EXE):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | offset in SHARE's DS of next lock table in list or 0000h |
| 02h | DWORD | offset in file of start of locked region |
| 06h | DWORD | offset in file of end of locked region |
| 0Ah | DWORD | pointer to System File Table entry for this file |
| 0Eh | WORD | PSP segment of lock's owner |
| ---DOS 5+ --- | | |
| 10h | WORD | lock type: (00h lock all, 01h lock writes only) |

**24-79 Format of DOS 3.1-3.3x, DR-DOS 5.0-6.0 system file tables and FCB tables:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to next file table (offset FFFFh if last) |
| 04h | WORD | number of files in this table |
| 06h | 35h bytes | per file |

| **Offset** | **Size** | **Description** |
|---|---|---|
| 00h | WORD | number of file handles referring to this file |
| 02h | WORD | file open mode (see Table 24-19)<br>bit 15 set if this file opened via FCB |
| 04h | BYTE | file attribute (see Table 24-21) |
| 05h | WORD | device info word (see Table 24-22)<br>bit 15 set if remote file<br>bit 14 set means do not set file date/time on closing<br>bit 12 set means don't inherit on EXEC<br>bits 5-0 drive number for disk files |
| 07h | DWORD | pointer to device driver header if character device<br>else pointer to DOS Drive Parameter Block (see Table 24-15) |
| 0Bh | WORD | starting cluster of file |
| 0Dh | WORD | file time in packed format (see Table 24-96)<br>not used for character devices in DR-DOS |
| 0Fh | WORD | file date in packed format (see Table 24-97)<br>not used for character devices in DR-DOS |
| 11h | DWORD | file size |
| ---system file table--- | | |
| 15h | DWORD | current offset in file (may be larger than size of file;<br>INT 21h Function 42h does not check new position) |
| ---FCB table--- | | |
| 15h | WORD | counter for last I/O to FCB |
| 17h | WORD | counter for last open of FCB<br>(these are separate to determine the times of the latest I/O and open) |

------

| | | |
|---|---|---|
| 19h | WORD | relative cluster within file of last cluster accessed |
| 1Bh | WORD | absolute cluster number of last cluster accessed<br>0000h if *file never read or written* |
| 1Dh | WORD | number of sector containing directory entry |
| 1Fh | BYTE | number of dir entry within sector (byte offset/32) |
| 20h | 11 BYTEs | filename in FCB format (no path/period, blank-padded) |
| 2Bh | DWORD | (SHARE.EXE) pointer to previous SFT sharing same file |
| 2Fh | WORD | (SHARE.EXE) network machine number which opened file (Windows Enhanced mode DOSMGR uses the virtual machine ID as the machine number; see INT 2Fh Function 1683h in Chapter 43) |
| 31h | WORD | PSP segment of file's owner (see Table 24-11) (first three entries for AUX/CON/PRN contain segment of IO.SYS startup code) |
| 33h | WORD | offset within SHARE code segment of sharing record (see Table 24-77)<br>0000h = none |

**Note:** The DOS 2.x and 3.0 system file table formats have been omitted due to lack of space; they are described in *Network Interrupts* and *Undocumented DOS*.

**24-80 Format of DOS 4.0-6.0 system file tables and FCB tables:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to next file table (offset FFFFh if last) |
| 04h | WORD | number of files in this table |
| 06h | 3Bh bytes per file | |

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of file handles referring to this file<br>FFFFh if in use but not referenced |
| 02h | WORD | file open mode (see Table 24-19)<br>bit 15 set if this file opened via FCB |
| 04h | BYTE | file attribute (see Table 24-21) |
| 05h | WORD | device info word (see also Table 24-22)<br>bit 15 set if remote file<br>bit 14 set means do not set file date/time on closing<br>bit 13 set if named pipe<br>bit 12 set if no inherit<br>bit 11 set if network spooler<br>bit 7 set if device, clear if file (only if local)<br>bits 6-0 as for Function 4400h |
| 07h | DWORD | pointer to device driver header if character device<br>else pointer to DOS Drive Parameter Block (see Table 24-15) or REDIR data |
| 0Bh | WORD | starting cluster of file (local files only) |
| 0Dh | WORD | file time in packed format (see Table 24-96) |
| 0Fh | WORD | file date in packed format (see Table 24-97) |
| 11h | DWORD | file size |
| 15h | DWORD | current offset in file (SFT)<br>LRU counters (FCB table, two WORDs) |

| | | |
|---|---|---|
| **---local file---** | | |
| 19h | WORD | relative cluster within file of last cluster accessed |
| 1Bh | DWORD | number of sector containing directory entry |
| 1Fh | BYTE | number of dir entry within sector (byte offset/32) |
| **---network redirector---** | | |
| 19h | DWORD | pointer to REDIRIFS record |
| 1Dh | 3 BYTEs | *unknown* |
| ------ | | |
| 20h | 11 BYTEs | filename in FCB format (no path/period, blank-padded) |
| 2Bh | DWORD | (SHARE.EXE) pointer to previous SFT sharing same file |
| 2Fh | WORD | (SHARE.EXE) network machine number which opened file (Windows Enhanced mode DOSMGR uses the virtual machine ID as the machine number; see INT 2Fh Function 1683h in Chapter 43) |
| 31h | WORD | PSP segment of file's owner (see Table 24-11) (first three entries for AUX/CON/PRN contain segment of IO.SYS startup code) |
| 33h | WORD | offset within SHARE.EXE code segment of sharing record (see Table 24-77)<br>0000h = none |
| 35h | WORD | (local) absolute cluster number of last cluster accessed<br>(redirector) *unknown* |
| 37h | DWORD | pointer to IFS driver for file, 0000000h if native DOS |

**24-81 Format of current directory structure (CDS) (array, LASTDRIVE entries):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 67 BYTEs | ASCIZ path in form X:\PATH (local) or \\MACH\PATH (network) |
| 43h | WORD | drive attributes (see also note below and Function 5F07h)<br>bit 15: uses network redirector } invalid if 00, installable file system if 11<br>bit 14: physical drive }<br>bit 13: JOIN'ed \ path above is true path that would be<br>bit 12: SUBST'ed / needed if not under SUBST or JOIN<br>bit 7: remote drive hidden from redirector's assign-list and exempt from network connection make/break commands; set for CD-ROM drives |
| 45h | DWORD | pointer to Drive Parameter Block for drive (see Table 24-15) |
| **---local drives---** | | |
| 49h | WORD | starting cluster of current directory<br>0000h = root, FFFFh = never accessed |
| 4Bh | WORD | *apparently always FFFFh* |
| 4Dh | WORD | *apparently always FFFFh* |
| **---network drives---** | | |
| 49h | DWORD | pointer to redirector or REDIRIFS record, or FFFFh:FFFFh<br>(DOS 4 only) available for use by IFS driver |
| 4Dh | WORD | stored user data from INT 21h Function 5F03h |
| ------ | | |
| 4Fh | WORD | offset in current directory path of backslash corresponding to root directory for drive this value specifies how many characters to hide from the "CHDIR" and "GETDIR" calls; normally set to 2 to hide the drive letter and colon, SUBST, JOIN, and networks change it so that only the appropriate portion of the true path is visible to the user. |

| | | |
|---|---|---|
| ---**DOS 4+** --- | | |
| 51h | BYTE | (DOS 4 only, remote drives) device type |
| | | 04h network drive |
| 52h | DWORD | pointer to IFS driver (DOS 4) or redirector block (DOS 5+) for this drive, |
| | | 00000000h if native DOS |
| 56h | WORD | available for use by IFS driver |

**Notes:** The path for invalid drives is normally set to X:\, but may be empty after JOIN x: /D in DR-DOS 5.0 or NET USE x: /D in older LAN versions.

Normally, only one of bits 13&12 may be set together with bit 14, but DR-DOS 5.0 uses other combinations for bits 15-12 (see Table 24-82).

### 24-82 Format of DR-DOS 5.0-6.0 current directory structure entry (array):

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 67 BYTEs | ASCIZ pathname of actual root directory for this logical drive |
| 43h | WORD | drive attributes |
| | | 1000h SUBSTed drive |
| | | *3000h JOINed drive* |
| | | 4000h physical drive |
| | | 5000h ASSIGNed drive |
| | | 7000h JOINed drive |
| | | 8000h network drive |
| 45h | BYTE | physical drive number (0=A:) if this logical drive is valid |
| 46h | BYTE | *apparently flags for JOIN and ASSIGN* |
| 47h | WORD | cluster number of start of parent directory (0000h = root) |
| 49h | WORD | entry number of current directory in parent directory |
| 4Bh | WORD | cluster number of start of current directory |
| 4Dh | WORD | used for media change detection (details not available) |
| 4Fh | WORD | cluster number of SUBST/JOIN "root" directory |
| | | 0000h if physical root directory |

### 24-83 Format of device driver header:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to next driver, offset=FFFFh if last driver |
| 04h | WORD | device attributes (see Table 24-84) |
| 06h | WORD | device strategy entry point |
| | | call with ES:BX -> request header (see Table 24-123) |
| 08h | WORD | device interrupt entry point |
| ---**character device**--- | | |
| 0Ah | 8 BYTEs | blank-padded character device name |
| ---**block device**--- | | |
| 0Ah | BYTE | number of subunits (drives) supported by driver |
| 0Bh | 7 BYTEs | unused |
| --- | | |
| 12h | WORD | (CD-ROM driver) reserved, must be 0000h |
| | | appears to be another device chain |
| 14h | BYTE | (CD-ROM driver) drive letter (must initially be 00h) |
| 15h | BYTE | (CD-ROM driver) number of units |
| 16h | 6 BYTEs | (CD-ROM driver) signature 'MSCDnn' where 'nn' is version (currently '00') |

### 24-84 Bitfields for device attributes:

| | |
|---|---|
| bit 15 | device type (clear = block device, set = character device) |
| bit 14 | IOCTL supported (see Functions 4402h to 4405h) |
| bit 13 | (block) non-IBM format<br>(DOS 3+ character) output until busy supported |
| bit 12 | (block) network/remote device |
| bit 11 | (DOS 3+) OPEN/CLOSE/RemMedia calls supported |
| bit 10 | reserved |
| bit 9 | (block) *direct I/O not allowed* (set by DOS 3.3 DRIVER.SYS) |
| bit 8 | (block) *unknown* (set by DOS 3.3 DRIVER.SYS for "new" drives) |
| bit 7 | (DOS 5+) Generic IOCTL check call supported (command 19h)<br>(see Functions 4410h and 4411h) |
| bit 6 | (DOS 3.2+) Generic IOCTL call supported (command 13h)<br>(see Functions 440Ch and 440Dh) |
| bit 5 | reserved |
| bit 4 | (character) device is special (supports fast console output INT 29h ) |
| bit 2 | (character) device is NUL |
| bit 1 | (DOS 3.31+ block) driver supports 32-bit sector addressing<br>(character) device is standard output |
| bit 0 | (character) device is standard input |

**Note:** For European MS-DOS 4.0, bit 11 also indicates that bits 8-6 contain a version code (000 = DOS 3.0,3.1; 001 = DOS 3.2, 010 = European DOS 4.0).

### 24-85 Format of CLOCK$ transfer record:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of days since 1-Jan-1980 |
| 02h | BYTE | minutes |
| 03h | BYTE | hours |
| 04h | BYTE | hundredths of second |
| 05h | BYTE | seconds |

### 24-86 Format of DOS 2.x disk buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to next disk buffer, offset = FFFFh if last least-recently used buffer is first in chain |
| 04h | BYTE | drive (0=A, 1=B, etc.), FFh if not in use |
| 05h | 3 BYTEs | *unused* (seems always to be 00h 00h 01h) |
| 08h | WORD | logical sector number |
| 0Ah | BYTE | number of copies to write (1 for non-FAT sectors) |
| 0Bh | BYTE | sector offset between copies if multiple copies to be written |
| 0Ch | DWORD | pointer to DOS Drive Parameter Block (see Table 24-15) |
| 10h | | buffered data |

### 24-87 Format of DOS 3.x disk buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to next disk buffer, offset = FFFFh if last least-recently used buffer is first in chain |
| 04h | BYTE | drive (0=A,1=B, etc.), FFh if not in use |
| 05h | BYTE | buffer flags (see Table 24-92) |
| 06h | WORD | logical sector number |
| 08h | BYTE | number of copies to write (1 for non-FAT sectors) |
| 09h | BYTE | sector offset between copies if multiple copies to be written |

| | | |
|---|---|---|
| 0Ah | DWORD | pointer to DOS Drive Parameter Block (see Table 24-15) |
| 0Eh | WORD | *unused* (almost always 0) |
| 10h | | buffered data |

### 24-88 Format of DOS 4.00 (pre UR 25066) disk buffer info:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to array of disk buffer hash chain heads (see Table 24-90) |
| 04h | WORD | number of disk buffer hash chains (referred to as NDBCH below) |
| 06h | DWORD | pointer to lookahead buffer, zero if not present |
| 0Ah | WORD | number of lookahead sectors, else zero (the y in BUFFERS=x,y) |
| 0Ch | BYTE | 00h if buffers in EMS (/X), FFh if not |
| 0Dh | WORD | EMS handle for buffers, zero if not in EMS |
| 0Fh | WORD | EMS physical page number used for buffers (usually 255) |
| 11h | WORD | *apparently always 0001h* |
| 13h | WORD | segment of EMS physical page frame |
| 15h | WORD | *apparently always zero* |
| 17h | 4 WORDs | *EMS partial page mapping information* |

### 24-89 Format of DOS 4.01 (from UR 25066 Corrective Services Disk on) disk buffer info:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to array of disk buffer hash chain heads (see Table 24-90) |
| 04h | WORD | number of disk buffer hash chains (referred to as NDBCH below) |
| 06h | DWORD | pointer to lookahead buffer, zero if not present |
| 0Ah | WORD | number of lookahead sectors, else zero (the y in BUFFERS=x,y) |
| 0Ch | BYTE | 01h, possibly to distinguish from pre-UR 25066 format |
| 0Dh | WORD | *EMS segment for BUFFERS* (only with /XD) |
| 0Fh | WORD | *EMS physical page number of EMS segment above* (only with /XD) |
| 11h | WORD | *EMS segment for unknown item* (only with /XD) |
| 13h | WORD | *EMS physical page number of above* (only with /XD) |
| 15h | BYTE | *number of EMS page frames present* (only with /XD) |
| 16h | WORD | segment of one-sector workspace buffer allocated in main memory if BUFFERS/XS or /XD options in effect, possibly to avoid DMA into EMS |
| 18h | WORD | EMS handle for buffers, zero if not in EMS |
| 1Ah | WORD | EMS physical page number used for buffers (usually 255) |
| 1Ch | WORD | *apparently always 0001h* |
| 1Eh | WORD | segment of EMS physical page frame |
| 20h | WORD | *apparently always zero* |
| 22h | BYTE | 00h if /XS, 01h if /XD, FFh if BUFFERS not in EMS |

### 24-90 Format of DOS 4.x disk buffer hash chain head (array, one entry per chain):

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | EMS logical page number in which chain is resident, -1 if not in EMS |
| 02h | DWORD | pointer to least recently used buffer header. All buffers on this chain are in the same segment. |
| 06h | BYTE | number of dirty buffers on this chain |
| 07h | BYTE | reserved (00h) |

**Notes:** Buffered disk sectors are assigned to chain N where N is the sector's address modulo the number of disk buffer chain heads (NDBCH), 0 <= N <= NDBCH-1. Each chain resides completely within one EMS page.

This structure is in main memory even if buffers are in EMS.

### 24-91 Format of DOS 4.0-6.0 disk buffer:

| Offset | Size | Description |
|---|---|---|
| 00h | WORD | forward ptr, offset only, to next least recently used buffer |
| 02h | WORD | backward ptr, offset only |
| 04h | BYTE | drive (0=A,1=B, etc.) if bit 7 clear<br>SFT index if bit 7 set<br>FFh if not in use |
| 05h | BYTE | buffer flags (see Table 24-92) |
| 06h | DWORD | logical sector number (local buffers only) |
| 0Ah | BYTE | number of copies to write<br>for FAT sectors, same as number of FATs<br>for data and directory sectors, usually 1 |
| 0Bh | WORD | offset in sectors between copies to write for FAT sectors |
| 0Dh | DWORD | pointer to DOS Drive Parameter Block (see Table 24-15) |
| 11h | WORD | size of data in buffer if remote buffer (see flags above) |
| 13h | BYTE | reserved (padding) |
| 14h | | buffered data |

**Note:** For DOS 4.x, all buffered sectors which have the same hash value computed as the sum of high and low words of the logical sector number divided by the number of disk buffer chains) are on the same doubly-linked circular chain; for DOS 5+, only a single circular chain exists.

The links consist of offset addresses only, the segment being the same for all buffers in the chain.

### 24-92 Bitfields for buffer flags:

| | |
|---|---|
| bit 7 | remote buffer |
| bit 6 | buffer dirty |
| bit 5 | buffer has been referenced (reserved in DOS 5+) |
| bit 4 | search data buffer (only valid if remote buffer) |
| bit 3 | sector in data area |
| bit 2 | sector in a directory, either root or subdirectory |
| bit 1 | sector in FAT |
| bit 0 | reserved |

### 24-93 Format of DOS 5.0-6.0 disk buffer info:

| Offset | Size | Description |
|---|---|---|
| 00h | DWORD | pointer to least-recently-used buffer header (may be in HMA) (see Table 24-91) |
| 04h | WORD | number of dirty disk buffers |
| 06h | DWORD | pointer to lookahead buffer, zero if not present |
| 0Ah | WORD | number of lookahead sectors, else zero (the y in BUFFERS=x,y) |
| 0Ch | BYTE | buffer location<br>00h base memory, no workspace buffer<br>01h HMA, workspace buffer in base memory |
| 0Dh | DWORD | pointer to one-segment workspace buffer in base memory |
| 11h | 3 BYTEs | unused |
| 14h | WORD | *unknown* |
| 16h | BYTE | flag: INT 24h fail while making an I/O status call |
| 17h | BYTE | temp storage for user memory allocation strategy during EXEC |
| 18h | BYTE | counter: number of INT 21h calls for which A20 is off |
| 19h | BYTE | bit flags<br>bit 0: *unknown* |

| | | |
|---|---|---|
| | | bit 1: SWITCHES=/W specified in CONFIG.SYS (don't load WINA20.SYS when MS Windows 3.0 starts) |
| | | bit 2: in EXEC state (INT 21h Function 4B05h) |
| 1Ah | WORD | offset of unpack code start (used only during INT 21h Function 4B05h) |
| 1Ch | BYTE | bit 0 set if UMB MCB chain linked to normal MCB chain |
| 1Dh | WORD | minimum paragraphs of memory required by program being EXECed |
| 1Fh | WORD | segment of first MCB in upper memory blocks or FFFFh if DOS memory chain in base 640K only (first UMB MCB usually at 9FFFh, locking out video memory with a DOS-owned memory block) |
| 21h | WORD | paragraph from which to start scanning during memory allocation |

### 24-94 Format of one item in DOS 4+ list of special program names:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | length of name (00h = end of list) |
| 01h | N BYTEs | name in format name.ext |
| N | 2 BYTEs | DOS version to return for program (major,minor) (see Function 30h, INT 2Fh Function 122Fh) |
| ---**DOS 4 only**--- | | |
| N+2 | BYTE | number of times to return fake version number (FFh = always) |

**Note:** If the name of the executable for the program making the DOS "get version" call matches one of the names in this list, DOS returns the specified version rather than the true version number.

## INT 21h Function 53h

*DOS 2+ internal*

### *TRANSLATE BIOS PARAMETER BLOCK TO DRIVE PARAMETER BLOCK*

**Registers at call:**
AH = 53h
DS:SI -> BIOS Parameter Block (see Table 24-95)
ES:BP -> buffer for Drive Parameter Block (see Table 24-15)

**Return Registers:**
ES:BP buffer filled

**Details:** For DOS 3+, the cluster at which to start searching is set to 0000h and the number of free clusters is set to FFFFh (unknown).

### 24-95 Format of BIOS Parameter Block:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of bytes per sector |
| 02h | BYTE | number of sectors per cluster |
| 03h | WORD | number of reserved sectors at start of disk |
| 05h | BYTE | number of FATs |
| 06h | WORD | number of entries in root directory |
| 08h | WORD | total number of sectors<br>for DOS 4+, set to zero if partition >32M, then set DWORD at 15h to actual number of sectors |
| 0Ah | BYTE | media ID byte |
| 0Bh | WORD | number of sectors per FAT |
| ---**DOS 3+**--- | | |
| 0Dh | WORD | number of sectors per track |
| 0Fh | WORD | number of heads |
| 11h | DWORD | number of hidden sectors |
| 15h | 11 BYTEs | reserved |

**---DOS 4+ ---**

| | | |
|---|---|---|
| 15h | DWORD | total number of sectors if word at 08h contains zero |
| 19h | 6 BYTEs | *unknown* |
| 1Fh | WORD | number of cylinders |
| 21h | BYTE | device type |
| 22h | WORD | device attributes (removable or not, etc.) |

**---DR-DOS 5+ ---**

| | | |
|---|---|---|
| 15h | DWORD | total number of sectors if word at 08h contains zero |
| 19h | 6 BYTEs | reserved |

**---European MS-DOS 4.00---**

| | | |
|---|---|---|
| 15h | DWORD | total number of sectors if word at 08h contains zero (however, this DOS does not actually implement >32M partitions) |

### INT 21h Function 54h

*DOS 2+*

#### *GET VERIFY FLAG*

**Registers at call:**
AH = 54h

**Return Registers:**
AL = verify flag
  00h off
  01h on (all disk writes verified after writing)

**See also:** Function 2Eh

### INT 21h Function 55h

*DOS 2+ internal*

#### *CREATE CHILD PSP*

**Registers at call:**
AH = 55h
DX = segment at which to create new PSP
SI = (DOS 3+) value to place in memory size field at DX:[0002h]

**Return Registers:**
CF clear if successful

**Details:** This function creates a "child" PSP rather than making an exact copy of the current PSP; the new PSP's parent pointer is set to the current PSP and the reference count for each inherited file is incremented.

DOS 2+ sets the current PSP to DX.

DOS 3+ marks "no inherit" file handles as closed in the child PSP.

**See also:** Functions 26h and 50h

### INT 21h Function 56h

*DOS 2+*

#### *"RENAME" - RENAME FILE*

**Registers at call:**
AH = 56h
DS:DX -> ASCIZ filename of existing file (no wildcards, but see below)
ES:DI -> ASCIZ new filename (no wildcards)
CL = attribute mask (server call only, see below)

**Return Registers:**
CF clear if successful
CF set on error
  AX = error code (02h,03h,05h,11h) (see Table 24-2)

**Details:** This function allows files to be moved between directories on the same logical volume. It does not set the archive attribute (see Table 24-21), which results in incremental backups not backing up the file under its new name.

Open files should not be renamed.

DOS 3+ allows renaming of directories.

DOS 3.1+ allows wildcards if invoked via Function 5D00h, in which case error 12h (no more files) is returned on success, and both source and destination specs must be canonical (as returned by Function 60h). Wildcards in the destination are replaced by the corresponding character of each source file being renamed. Under DOS 3.x, the call will fail if the destination wildcard is *.* or equivalent; under DR-DOS 5.0, the call will fail if any wildcards are used. When invoked via Function 5D00h, only those files matching the attribute mask in CL are renamed.

Under the FlashTek X-32 DOS extender, the old-name pointer is in DS:EDX and the new-name pointer is in ES:EDI (DS must equal ES)

**BUG:** Under DR-DOS 3.41, this function will generate a new directory entry with the new name (including any wildcards) which can only be removed with a sector editor when invoked via Function 5D00h.

**See also:** Functions 17h, 4301h, 5D00h, and 60h

---

## INT 21h Function 5700h

*DOS 2+*

### *GET FILE'S DATE AND TIME*

**Registers at call:**

AX = 5700h

BX = file handle

**Return Registers:**

CF clear if successful

CX = file's time (see Table 24-96)

DX = file's date (see Table 24-97)

CF set on error

AX = error code (01h,06h) (see Table 24-2)

**Details:** Under DR-DOS 3.41 and 5.0, this function returns 0 (no date/time) for character devices; MS-DOS returns the date and time of opening.

**See also:** Function 5701h

**24-96 Bitfields for file time:**

| | |
|---|---|
| bits 15-11 | hours (0-23) |
| bits 10-5 | minutes |
| bits 4-0 | seconds/2 |

**24-97 Bitfields for file date:**

| | |
|---|---|
| bits 15-9 | year - 1980 |
| bits 8-5 | month |
| bits 4-0 | day |

---

## INT 21h Function 5701h

*DOS 2+*

### *SET FILE'S DATE AND TIME*

**Registers at call:**

AX = 5701h

BX = file handle

CX = new time (see Table 24-96)

DX = new date (see Table 24-97)

**Return Registers:**

CF clear if successful

CF set on error

AX = error code (01h,06h) (see Table 24-2)

**See also:** Function 5700h

---

## INT 21h Function 5702h

*DOS 4.x only*

### *GET EXTENDED ATTRIBUTES FOR FILE*

**Purpose:** Get the current value of one or more extended attributes.

**Registers at call:**

AX = 5702h

BX = file handle

CX = size of result buffer or 0000h

DS:SI -> EAP list (see Table 24-98)

ES:DI -> buffer for returned EAV list (see Table 24-99)

**Return Registers:**

CF clear if successful

CX = size of returned data

CF set on error

AX = error code (see Table 24-2)

**Details:** If CX=0000h on entry, ES:DI is ignored and no data is actually returned, only the amount of data which is available.

The default DOS 4 behavior is to return a single word of 0000h (no structures) in the result buffer if CX>=0002h on entry.

**Conflicts:** OS/2 (Chapter 26)

**See also:** Functions 5703h and 5704h, OS/2 Function 6Eh (Chapter 26), Network Redirector INT 2Fh Function 112Dh (NI)

### 24-98 Format of EAP (extended attribute properties) list:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of EAP structures following |
| 02h | varies | array of EAP structures (see Table 24-99) |

### 24-99 Format of EAP (extended attribute property) structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | attribute type<br>01h boolean (either 00h or 01h)<br>02h number (BYTE, WORD, or DWORD)<br>03h string<br>04h date stamp<br>05h time stamp |
| 01h | WORD | EAP flags (see Table 24-100) |
| 03h | BYTE | size of reference string (name) |
| 04h | N BYTEs | reference string |

### 24-100 Bitfields for EAP flags:

| | |
|---|---|
| bit 12 | unchangeable |
| bit 13 | ignore |
| bit 14 | unchangeable |
| bit 15 | used by COMMAND.COM for code page, but not understood by ATTRIB |

### 24-101 Format of EAV (extended attribute value) list:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of EAV structures following |
| 02h | varies | array of Extended Attribute Value structures |

### 24-102 Format of Extended Attribute Value structures:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 4 BYTEs | *unknown* |
| 04h | BYTE | size of reference string |
| 05h | WORD | size of value |
| 07h | varies | reference string |
| | varies | value |

## INT 21h Function 5703h

*DOS 4.x only*

### *GET EXTENDED ATTRIBUTE PROPERTIES*

**Purpose:** Get a list of the extended attributes which are defined for the specified file.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5703h | CF clear if successful |
| BX = file handle | CX = size of returned data |
| CX = size of result buffer or 0000h | CF set on error |
| ES:DI -> result buffer | AX = error code (see Table 24-2) |
| | ES:DI -> zero word (DOS 4.0) if CX >= 2 on entry |

**Details:** If CX=0000h on entry, ES:DI is ignored and no data is actually returned, only the amount of data which is available.

The default DOS 4 behavior is to return a trivial EAP list consisting of the single word 0000h (no EAP structures) if CX>=0002h on entry.
**Conflicts:** OS/2 (Chapter 26)
**See also:** Functions 5702h and 5704h, OS/2 Function 6Eh (Chapter 26), Network Redirector INT 2Fh Function 112Dh (NI)

---

## INT 21h Function 5704h

*DOS 4.x only*

### *SET EXTENDED ATTRIBUTES*

**Registers at call:**
AX = 5704h
BX = file handle
ES:DI -> EAV list (see Table 24-101)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 24-2)

**Details:** The default DOS 4 behavior is to do nothing and return successfully.
**See also:** Functions 5702h and 5703h, Network Redirector INT 2Fh Function 112Dh (NI)

---

## INT 21h Functions 5800h and 5801h

*DOS 3+*

### *GET OR SET MEMORY ALLOCATION STRATEGY*

**Registers at call:**
AH = 58h
AL = subfunction
  00h get allocation strategy
  01h set allocation strategy
    BL = new allocation strategy (see Table 24-103)
    BH = 00h (DOS 5+)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (01h) (see Table 24-2)
AX = current strategy (see Table 24-103) if successful

**Details:** The Set subfunction accepts any value in BL for DOS 3.x and 4.x; 2 or greater means last fit. The Get subfunction returns the last value set.

Setting an allocation strategy involving high memory does not automatically link in the UMB memory chain; this must be done explicitly with Function 5803h in order to actually allocate high memory.

A program which changes the allocation strategy should restore it before terminating.

Toshiba MS-DOS 2.11 supports subfunctions 00h and 01h.

DR-DOS 3.41 reportedly reverses subfunctions 00h and 01h.

**See also:** Functions 48h, 49h, and 4Ah; DOSKEY INT 2Fh Function 4310h (Chapter 27), QEMM INT 67h Function 3Fh (Chapter 39)

**24-103 Memory allocation strategies:**

| | |
|---|---|
| 00h | low memory first fit |
| 01h | low memory best fit |
| 02h | low memory last fit |
| ---DOS 5+ --- | |
| 40h | high memory first fit |
| 41h | high memory best fit |
| 42h | high memory last fit |
| 80h | first fit, try high then low memory |
| 81h | best fit, try high then low memory |
| 82h | last fit, try high then low memory |

## INT 21h Functions 5802h and 5803h *DOS 5+*

### *GET OR SET UMB LINK STATE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 58h<br>AL = subfunction | CF clear if successful<br>CF set on error<br>AX = error code (01h) (see Table 24-2) |
| 02h get UMB link state | AL = state if successful<br>00h UMBs not part of DOS memory chain<br>01h UMBs in DOS memory chain |
| 03h set UMB link state<br>BX = new state<br>0000h remove UMBs from DOS memory chain<br>0001h add UMBs to DOS memory chain | |

**Details:** A program which changes the UMB link state should restore it before terminating.

## INT 21h Function 59h, Subfunction 0000h *DOS 3+*

### *GET EXTENDED ERROR INFORMATION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 59h<br>BX = 0000h | AX = extended error code (see Table 24-2)<br>BH = error class (see Table 24-104)<br>BL = recommended action (see Table 24-105)<br>CH = error locus (see Table 24-106)<br>ES:DI -> ASCIZ volume label of required disk if error 22h<br>CL, DX, SI, BP, and DS destroyed |

**Details:** Functions available under DOS 2.x map the true DOS 3+ error code into one supported under DOS 2.x. You should call this function to retrieve the true error code when an FCB or DOS 2.x call returns an error.

Under DR-DOS 5.0, this function does not use any of the DOS-internal stacks and may thus be called at any time.

**See also:** Function 5D0Ah, INT 2Fh Function 122Dh (below), European MS-DOS Function 59h Subfunction 0001h (Chapter 25)

**24-104 Values for Error Class:**

| | |
|---|---|
| 01h | out of resource (storage space or I/O channels) |
| 02h | temporary situation (file or record lock) |
| 03h | authorization (denied access) |
| 04h | internal (system software bug) |
| 05h | hardware failure |
| 06h | system failure (configuration file missing or incorrect) |
| 07h | application program error |
| 08h | not found |
| 09h | bad format |
| 0Ah | locked |
| 0Bh | media error |
| 0Ch | already exists |
| 0Dh | unknown |

### 24-105 Values for Suggested Action:

| | |
|---|---|
| 01h | retry |
| 02h | delayed retry |
| 03h | prompt user to reenter input |
| 04h | abort after cleanup |
| 05h | immediate abort |
| 06h | ignore |
| 07h | retry after user intervention |

### 24-106 Values for Error Locus:

| | |
|---|---|
| 01h | unknown or not appropriate |
| 02h | block device (disk error) |
| 03h | network related |
| 04h | serial device (timeout) |
| 05h | memory related |

---

## INT 21h Function 5Ah

*DOS 3+*

### *CREATE TEMPORARY FILE*

**Purpose:** This function creates a file with a unique name which must be explicitly deleted.

**Registers at call:**
AH = 5Ah
CX = file attribute (see Table 24-21)
DS:DX -> ASCIZ path ending with a '\' + 13 zero bytes to receive the generated filename

**Return Registers:**
CF clear if successful
AX = file handle opened for read/write in compatibility mode
DS:DX pathname extended with generated name for temporary file
CF set on error
AX = error code (03h,04h,05h) (see Table 24-2)

**Details:** Under the FlashTek X-32 DOS extender, the path pointer is in DS:EDX.

**BUGS:** COMPAQ DOS 3.31 hangs if the pathname is at offset 0000h in any segment; it apparently wraps around to the end of the segment.

MS-DOS 5.00 revisions A and B and PC-DOS 5.00 revision A reportedly hang the system if the specified path is the root directory and the root directory is full (no free directory entries).

**See also:** Functions 3Ch and 5Bh

---

## INT 21h Function 5Bh

*DOS 3+*

### *CREATE NEW FILE*

**Registers at call:**
AH = 5Bh
CX = file attribute (see Table 24-21)
DS:DX -> ASCIZ filename

**Return Registers:**
CF clear if successful
AX = file handle opened for read/write in compatibility mode
CF set on error
AX = error code (03h,04h,05h,50h) (see Table 24-2)

**Details:** Unlike Function 3Ch, this function will fail if the specified file exists rather than truncating it; this permits its use in creating semaphore files because it is an atomic "test and set" operation.

Under the FlashTek X-32 DOS extender, the filename pointer is in DS:EDX.

**See also:** Functions 3Ch and 5Ah

---

## INT 21h Function 5Ch

*DOS 3+*

### *"FLOCK" - RECORD LOCKING*

**Registers at call:**
AH = 5Ch
AL = subfunction

00h lock region of file
01h unlock region of file
BX = file handle
CX:DX = start offset of region within file
SI:DI = length of region in bytes

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (01h,06h,21h,24h) (see Table 24-2)

**Details:** An error is returned unless SHARE or a network is installed.

An unlock call must specify the same region as some prior lock call.

Locked regions become entirely inaccessible to other processes.

Duplicate handles created with Functions 45h or 46h inherit locks, but handles inherited by child processes (see Function 4Bh) do not.

Under DR-DOS 3.41 and 5.0, if a process opens a file without the no-inherit flag and then starts a child, any locks set by the parent are ignored, and the child will only get an error if it tries to lock an area previously locked by the parent process.

**See also:** Function 440Bh, Novell NetWare Functions BCh and BEh (NI), Network Redirector INT 2Fh Functions 110Ah and 110Bh (NI)

---

## INT 21h Function 5D00h
### *SERVER FUNCTION CALL*

*DOS 3.1+ internal (undocumented)*

**Registers at call:**
AX = 5D00h
DS:DX -> DOS parameter list (see Table 24-107)
DPL contains all register values for a call to INT 21h

**Return Registers:**
as appropriate for function being called

**Details:** This function does not check AH; out of range values will crash the system.

The specified INT 21h function executes using the given computer ID and process ID.

Sharing delay loops are skipped, a special sharing mode is enabled to handle FCBs opened across network, and wildcards are enabled for DELETE (Function 41h) and RENAME (Function 56h) under MS-DOS; under DR-DOS 3.41, wildcards corrupt the filesystem, and under DR-DOS 5.0, the call returns error code 03h.

An extra file attribute parameter is enabled for OPEN (Function 3Dh), DELETE (Function 41h), and RENAME (Function 56h).

Functions which take filenames require canonical names (as returned by Function 60h); this is apparently to prevent multi-hop file forwarding.

**See also:** Functions 3Dh, 41h, 56h, and 60h

**24-107 Format of DOS parameter list:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | AX |
| 02h | WORD | BX |
| 04h | WORD | CX |
| 06h | WORD | DX |
| 08h | WORD | SI |
| 0Ah | WORD | DI |
| 0Ch | WORD | DS |
| 0Eh | WORD | ES |
| 10h | WORD | reserved (0) |
| 12h | WORD | computer ID (0 = current system) |
| 14h | WORD | process ID (PSP segment on specified computer) |

**Note:** Under Windows Enhanced mode, the computer ID is normally the virtual machine ID (see INT 2Fh Function 1683h in Chapter 43), though this can reportedly be changed by setting UniqueDOSPSP= in SYSTEM.INI.

---

### INT 21h Function 5D01h
### *COMMIT ALL FILES FOR SPECIFIED COMPUTER/PROCESS*

*DOS 3.1+ internal*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5D01h | CF set on error |
| DS:DX -> DOS parameter list (see Table 24-107), only computer ID and process ID fields used | AX = error code (see Table 24-2) |
| | CF clear if successful |

**Details:** This function flushes buffers and updates directory entries for each file which has been modified; if a remote file, it calls INT 2Fh Function 1107h.

The computer ID and process ID are stored but ignored under DOS 3.3.

This function is not supported by DR-DOS 3.41 and 5.0; returns error code 01h.

**See also:** Functions 0Dh and 68h, Network Redirector INT 2Fh Function 1107h (NI)

---

### INT 21h Function 5D06h
### *GET ADDRESS OF DOS SWAPPABLE DATA AREA*

*DOS 3.0+ internal*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5D06h | CF set on error |
| | AX = error code (see Table 24-2) |
| | CF clear if successful |
| | DS:SI -> nonreentrant data area (see Tables 24-108 and 24-110) |
| | CX = size in bytes of area which must be swapped while in DOS |
| | DX = size in bytes of area which must always be swapped |

**Details:** The Critical Error flag is used in conjunction with the InDOS flag (see Function 34h) to determine when it is safe to enter DOS from a TSR. Setting this flag allows use of Functions 50h and 51h from INT 28h under DOS 2.x by forcing use of the correct stack.

Swapping the data area allows reentering DOS unless DOS is in a critical section delimited by INT 2Ah Function 80h and INT 2Ah Function 81h,82h (see *Network Interrupts* or *Undocumented DOS*).

Under DOS 4.0x, Function 5D0Bh should be used instead of this function.

SHARE and other DOS utilities consult the byte at offset 04h in the DOS data segment (see INT 2Fh Function 1203h) to determine the SDA format in use: 00h = DOS 3.x, 01h = DOS 4.0-6.0, other = error.

DR-DOS 3.41+ supports this function, but the SDA format beyond the first 18h bytes is completely different from MS-DOS.

**See also:** Function 5D0Bh; Network INT 2Ah Functions 80h, 81h, and 82h (NI)

**24-108 Format of DOS 3.10-3.30 Swappable Data Area:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -11 | 5 WORDs | zero-terminated list of offsets which need to be patched to enable critical-section calls (see INT 2Ah Function 80h in *Network Interrupts* or *Undocumented DOS*)<br>(not actually part of the SDA) |
| -1 | BYTE | unused padding |
| 00h | BYTE | critical error flag ("ErrorMode") |
| 01h | BYTE | InDOS flag (count of active INT 21h calls) |

| | | |
|---|---|---|
| 02h | BYTE | drive on which current critical error occurred, or FFh (DR-DOS sets to drive number during INT 24h, 00h otherwise) |
| 03h | BYTE | locus of last error |
| 04h | WORD | extended error code of last error |
| 06h | BYTE | suggested action for last error |
| 07h | BYTE | class of last error |
| 08h | DWORD | ES:DI pointer for last error |
| 0Ch | DWORD | current DTA |
| 10h | WORD | current PSP |
| 12h | WORD | stores SP across an INT 23h |
| 14h | WORD | return code from last process termination (cleared after reading with Function 4Dh) |
| 16h | BYTE | current drive |
| 17h | BYTE | extended break flag |

**---remainder need only be swapped if in DOS---**

| | | |
|---|---|---|
| 1Ah | WORD | PSP segment for sharing/network |
| 1Ch | WORD | network machine number for sharing/network (0000h = us) |
| 37h | BYTE | flag: if nonzero, INT 24h Abort turned into INT 24h Fail (set only during process termination) |
| 84h | 3 BYTEs | 24-bit user number (see Function 30h) |
| 87h | BYTE | OEM number (see Table 24-14) |
| 92h | 128 BYTEs | buffer for filename |
| 112h | 128 BYTEs | buffer for filename |
| 192h | 21 BYTEs | findfirst/findnext search data block (see Table 24-68) |
| 1A7h | 32 BYTEs | directory entry for found file (see Table 24-7) |
| 1C7h | 81 BYTEs | copy of current directory structure for drive being accessed |
| 23Fh | BYTE | flag indicating how DOS function was invoked (00h = direct INT 20/INT 21h, FFh = server call Function 5D00h) |
| 262h | BYTE | Media ID byte returned by Functions 1Bh and 1Ch |
| 26Ch | DWORD | pointer to current directory structure for drive being accessed |
| 27Ch | WORD | offset in DOS DS of first filename argument |
| 27Eh | WORD | offset in DOS DS of second filename argument |
| 2EDh | 331 BYTEs | critical error stack |
| 438h | 384 BYTEs | disk stack (functions greater than 0Ch, INT 25h, INT 26h) |
| 5B8h | 384 BYTEs | character I/O stack (functions 01h through 0Ch) |

**---DOS 3.2,3.3x only---**

| | | |
|---|---|---|
| 738h | BYTE | device driver lookahead flag (usually printer) (see Function 64h) |
| 739h | BYTE | volume change flag |
| 73Ah | BYTE | flag: virtual open |

**Note:** For space reasons, most of the fields in the SDA have been omitted here. A full listing is available in *Network Interrupts* and *Undocumented DOS.*

## INT 21h Function 5D0Ah *DOS 3.1+*

### *SET EXTENDED ERROR INFORMATION*

| Registers at call: | Return Registers: |
|---|---|
| AX = 5D0Ah<br>DS:DX -> DOS parameter list (see Table 24-107) | none. The next call to Function 59h will return values from fields AX,BX,CX,DX,DI, and ES in the corresponding registers. |

**Details:** This function was undocumented prior to the release of MS-DOS 5.0.

The *MS-DOS Programmer's Reference* incorrectly states that this call was introduced in DOS 4, and fails to mention that the ERROR structure passed to this function is a DOS parameter list.
**BUG:** DR-DOS 3.41 and 5.0 read the value for ES from the DS field of the DPL; fortunately, MS-DOS ignores the DS field, allowing a generic routine which sets both DS and ES fields to the same value.
**See also:** Function 59h

---

## INT 21h Function 5D0Bh
## *GET DOS SWAPPABLE DATA AREAS*

*DOS 4.x only internal*
*(undocumented)*

**Registers at call:**
AX = 5D0Bh

**Return Registers:**
CF set on error
AX = error code (see Table 24-2)
CF clear if successful
DS:SI -> swappable data area list (see Table 24-109)

**Details:** Copying and restoring the swappable data areas allows DOS to be reentered unless it is in a critical section delimited by calls to INT 2Ah Function 80h and INT 2Ah Functions 81h,82h (see *Network Interrupts* or *Undocumented DOS*).

SHARE and other DOS utilities consult the byte at offset 04h in the DOS data segment (see INT 2Fh Function 1203h) to determine the SDA format in use: 00h = DOS 3.x, 01h = DOS 4.0-6.0, other = error.

DOS 5+ use the SDA format listed below, but revert back to the DOS 3.x call for finding the SDA (see Function 5D06h).
**See also:** Function 5D06h; INT 2Fh Function 1203h; Network INT 2Ah Functions 80h, 81h, and 82h (NI)

### 24-109 Format of DOS 4.x swappable data area list:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | count of data areas |
| 02h | N BYTEs | "count" copies of data area record |

| Offset | Size | Description |
|---|---|---|
| 00h | DWORD | address |
| 04h | WORD | length and type<br>bit 15 set if swap always, clear if swap in DOS<br>bits 14-0: length in bytes |

### 24-110 Format of DOS 4.0-6.0 swappable data area:

| *Offset* | *Size* | *Description* |
|---|---|---|
| -11 | 5 WORDs | zero-terminated list of offsets which need to be patched to enable critical-section calls (see INT 2Ah Function 80h in *Network Interrupts* or *Undocumented DOS*)<br>(not actually part of the SDA; all offsets are 0D0Ch, but this list is still present for DOS 3.x compatibility) |
| -1 | BYTE | unused padding |
| 00h | BYTE | critical error flag ("ErrorMode") |
| 01h | BYTE | InDOS flag (count of active INT 21h calls) |
| 02h | BYTE | drive on which current critical error occurred or FFh |
| 03h | BYTE | locus of last error |
| 04h | WORD | extended error code of last error |
| 06h | BYTE | suggested action for last error |
| 07h | BYTE | class of last error |
| 08h | DWORD | ES:DI pointer for last error |

| | | |
|---|---|---|
| 0Ch | DWORD | current DTA |
| 10h | WORD | current PSP |
| 12h | WORD | stores SP across an INT 23h |
| 14h | WORD | return code from last process termination (zeroed after reading with Function 4Dh) |
| 16h | BYTE | current drive |
| 17h | BYTE | extended break flag |
| 18h | BYTE | flag: code page switching |
| 19h | BYTE | flag: copy of previous byte in case of INT 24h Abort |
| **---remainder need only be swapped if in DOS---** | | |
| 1Ch | WORD | PSP segment for sharing/network |
| 1Eh | WORD | network machine number for sharing/network (0000h = us) |
| 39h | BYTE | flag: abort currently in progress, turn INT 24h Abort into Fail |
| 92h 3 | BYTEs | 24-bit user number (see Function 30h) |
| 95h | BYTE | OEM number (see Table 24-14) |
| 9Eh | 128 BYTEs | buffer for filename |
| 11Eh | 128 BYTEs | buffer for filename |
| 19Eh | 21 BYTEs | findfirst/findnext search data block (see Table 24-68) |
| 1B3h | 32 BYTEs | directory entry for found file (see Table 24-11) |
| 1D3h | 88 BYTEs | copy of current directory structure for drive being accessed |
| 252h | BYTE | flag indicating how DOS function was invoked<br>(00h = direct INT 20/INT 21h, FFh = server call Function 5D00h) |
| 278h | BYTE | Media ID byte returned by Functions 1Bh and 1Ch |
| 282h | DWORD | pointer to current directory structure for drive being accessed |
| 292h | WORD | offset in DOS DS of first filename argument |
| 294h | WORD | offset in DOS DS of second filename argument |
| 2F3h | DWORD | pointer to user-supplied filename |
| 335h | 331 BYTEs | critical error stack |
| 480h | 384 BYTEs | disk stack (functions greater than 0Ch, INT 25h, INT 26h) |
| 600h | 384 BYTEs | character I/O stack (functions 01h through 0Ch) |
| 780h | BYTE | device driver lookahead flag (usually printer) (see Function 64h) |
| 781h | BYTE | volume change flag |
| 782h | BYTE | flag: virtual open |
| 784h | 4 WORDs | *unknown* |

**Note:** For space reasons, most of the fields in the SDA have been omitted here. A full listing is available in *Network Interrupts* and *Undocumented DOS.*

## INT 21h Functions 5Exxh *DOS 3.1+*

***NETWORK FUNCTIONS—see*** *Network Interrupts*

## INT 21h Function 5F07h *DOS 5+*

***ENABLE DRIVE***

**Registers at call:**
AX = 5F07h
DL = drive number (0=A:)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (0Fh) (see Table 24-2)

**Details:** This function simply sets the "valid" bit in the drive's CDS.

**Conflicts:** STARLITE/Embedded DOS (NI)

**See also:** Functions 52h and 5F08h

---

DOS 5+

## INT 21h Function 5F08h

### *DISABLE DRIVE*

**Registers at call:**
AX = 5F08h
DL = drive number (0=A:)

**Return Registers:**
CF clear if successful
CF set on error
    AX = error code (0Fh) (see Table 24-2)

**Details:** This function simply clears the "valid" bit in the drive's CDS.
**Conflicts:** STARLITE/Embedded DOS (NI)
**See also:** Functions 52h and 5F07h

---

DOS 3.0+

## INT 21h Function 60h

### *"TRUENAME" - CANONICALIZE FILENAME OR PATH*

**Purpose:** Determine the canonical name of the specified filename or path, corresponding to the undocumented TRUENAME command in COMMAND.COM.

**Registers at call:**
AH = 60h
DS:SI -> ASCIZ filename or path
ES:DI -> 128-byte buffer for
  canonicalized name

**Return Registers:**
CF set on error
    AX = error code
        02h invalid component in directory
          path or drive letter only
        03h malformed path or invalid drive letter
          ES:DI buffer unchanged
CF clear if successful
    AH = 00h
    AL = destroyed (00h or 5Ch or last character
      of current directory on drive)
    buffer filled with qualified name of form
      D:\PATH\FILE.EXT or
      \\MACHINE\PATH\FILE.EXT

**Details:** The input path need not actually exist.

Letters are uppercased, forward slashes converted to backslashes, asterisks converted to appropriate number of question marks, and file and directory names are truncated to 8.3 if necessary. (DR-DOS 3.41 and 5.0 do not expand asterisks).

'.' and '..' in the path are resolved.

Filespecs on local drives always start with "d:", those on network drives always start with "\\".

If the path string is on a JOINed drive, the returned name is the one that would be needed if the drive were not JOINed; similarly for a SUBSTed, ASSIGNed, or network drive letter. Because of this, it is possible to get a qualified name that is not legal under the current combination of SUBSTs, ASSIGNs, JOINs, and network redirections.

Under DOS 3.3 through 6.00, a device name is translated differently if the device name does not have an explicit directory or the directory is \DEV (relative directory DEV from the root directory works correctly). In these cases, the returned string consists of the unchanged device name and extension appended to the string X:/ (forward slash instead of backward slash as in all other cases) where X is the default or explicit drive letter.

Functions which take pathnames require canonical paths if invoked via INT 21h Function 5D00h.

This function is supported by the OS/2 v1.1 compatibility box.

NetWare 2.1x does not support characters with the high bit set; early versions of NetWare 386 support such characters except in this call. In addition, NetWare returns error code 3 for the path "X:\"; one should use "X:\." instead.

For DOS 3.3-6.0, the input and output buffers may be the same, as the canonicalized name is built in an internal buffer and copied to the specified output buffer as the very last step. For DR

DOS 6.0, this function is not automatically called when on a network. Device drivers reportedly cannot make this call from their INIT function. Using the same pointer for both input and output buffers is not supported in the April 1992 and earlier versions of DR DOS.
**See also:** Function 5FB3h, INT 2Fh Function 1221h (below), Network Redirector INT 2Fh Function 1123h (NI)

---

### INT 21h Function 61h
*DOS 3+*

### *UNUSED*

**Registers at call:**
AH = 61h

**Return Registers:**
AL = 00h

**Details:** This function does nothing and returns immediately.
**Conflicts:** OS/2 (Chapter 26)

---

### INT 21h Function 62h
*DOS 3+*

### *GET CURRENT PSP ADDRESS*

**Registers at call:**
AH = 62h

**Return Registers:**
BX = segment of PSP for current process

**Details:** Under DOS 3+, this function does not use any of the DOS-internal stacks and may thus be called at any time, even during another INT 21h call.

The current PSP is not necessarily the caller's PSP.

This function is identical to the undocumented Function 51h.

**See also:** Functions 50h and 51h

---

### INT 21h Function 6300h
*DOS 2.25 only*

### *GET LEAD BYTE TABLE ADDRESS*

**Registers at call:**
AX = 6300h

**Return Registers:**
CF clear if successful
    DS:SI -> lead byte table (see Table 24-111)
CF set on error
    AX = error code (01h) (see Table 24-2)

**Details:** This function does not preserve any registers other than SS:SP.

The US version of MS-DOS 3.30 treats this as an unused function, setting AL=00h and returning immediately.

**See also:** Functions 07h, 08h, 0Bh, and 6301h

**24-111 Format of lead byte table entry:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | low/high ends of a range of leading byte of double-byte chars |
| 02h | 2 BYTEs | low/high ends of a range of leading byte of double-byte chars |
| | ... | |
| N | 2 BYTEs | 00h,00h end flag |

---

### INT 21h Function 6300h
*Far East DOS 3.2+*

### *GET DOUBLE BYTE CHARACTER SET LEAD-BYTE TABLE*

**Registers at call:**
AX = 6300h

**Return Registers:**
AL = error code
    00h successful
        DS:SI -> DBCS table (see Table 24-112)
        all other registers except CS:IP and
          SS:SP destroyed
    FFh not supported

**Details:** This function is probably identical to Function 6300h for DOS 2.25.

The US version of MS-DOS 3.30 treats this as an unused function, setting AL=00h and returning immediately. The US version of DOS 4.0+ accepts this function, but returns an empty list.

**See also:** Function 6300h above

**24-112 Format of DBCS table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | low/high ends of a range of leading byte of double-byte chars |
| 02h | 2 BYTEs | low/high ends of a range of leading byte of double-byte chars |
| | ... | |
| N | 2 BYTEs | 00h,00h end flag |

### INT 21h Function 6301h

*DOS 2.25, Far East DOS 3.2+*

### *SET KOREAN (HANGEUL) INPUT MODE*

**Registers at call:**

AX = 6301h
DL = new mode
    00h return only full characters on DOS keyboard input functions
    01h return partially-formed (interim) characters also

**Return Registers:**

AL = status
    00h successful
    FFh invalid mode

**See also:** Functions 07h, 08h, 0Bh, and 6302h

### INT 21h Function 6302h

*DOS 2.25, Far East DOS 3.2+*

### *GET KOREAN (HANGEUL) INPUT MODE*

**Registers at call:**

AX = 6302h

**Return Registers:**

AL = status
    00h successful
        DL = current input mode
            00h return only full characters
            01h return partial characters
    FFh not supported

**See also:** Functions 07h, 08h, 0Bh, and 6301h

### INT 21h Function 64h

*DOS 3.2+ internal*

### *SET DEVICE DRIVER LOOKAHEAD FLAG*

**Registers at call:**

AH = 64h
AL = flag
    00h (default) call device driver function 5 (non-destructive read) before INT 21h Functions 01h,08h,0Ah
    nonzero don't call driver function 5

**Return Registers:**

nothing (MS-DOS)
CF set, AX=*error code* (DR-DOS 5.0, which does not support this call)

**Details:** This function is called by DOS 3.3+ PRINT.COM.

Under MS-DOS, this function does not use any of the DOS-internal stacks and may thus be called at any time, even during another DOS call.

**Conflicts:** OS/2 (Chapter 26)

**See also:** Functions 01h, 08h, 0Ah, and 5D06h

## INT 21h Functions 6501h to 6507h

*DOS 3.3+*

### *GET EXTENDED COUNTRY INFORMATION*

**Registers at call:**
AH = 65h
AL = info ID
  01h get general internationalization info
  02h get pointer to uppercase table
  04h get pointer to filename uppercase table
  05h get pointer to filename terminator table
  06h get pointer to collating sequence table
  07h (DOS 4+) get pointer to Double-Byte Character Set table
BX = code page (FFFFh=global code page)
DX = country ID (FFFFh=current country)
ES:DI -> country information buffer (see Table 24-113)
CX = size of buffer (>= 5)

**Return Registers:**
CF set on error
  AX = error code (see Table 24-2)
CF clear if successful
  CX = size of country information returned
  ES:DI -> country information

**Details:** AL=05h appears to return the same information for all countries and codepages; it was undocumented prior to the release of MS-DOS 5.0.

NLSFUNC must be installed to get information for countries other than the default.

Subfunctions 02h and 04h are identical under OS/2.

**See also:** Function 38h, NLSFUNC INT 2Fh Functions 1401h and 1402h (Chapter 27), DR-DOS INT 2Fh Function 14FEh (Chapter 26)

**24-113 Format of country information:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | info ID |
| **---if info ID = 01h---** | | |
| 01h | WORD | size |
| 03h | WORD | country ID |
| 05h | WORD | code page |
| 07h | 34 BYTEs | country-dependent info (see Table 24-17) |
| **---if info ID = 02h---** | | |
| 01h | DWORD | pointer to uppercase table (see Table 24-114) |
| **---if info ID = 04h---** | | |
| 01h | DWORD | pointer to filename uppercase table (see Table 24-117) |
| **---if info ID = 05h---** | | |
| 01h | DWORD | pointer to filename character table (see Table 24-116) |
| **---if info ID = 06h---** | | |
| 01h | DWORD | pointer to collating table (see Table 24-115) |
| **---if info ID = 07h (DOS 4+)---** | | |
| 01h | DWORD | pointer to DBCS lead byte table (see Table 24-118) |

**24-114 Format of uppercase table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | table size |
| 02h | 128 BYTEs | uppercase equivalents (if any) of characters 80h to FFh |

### 24-115 Format of collating table:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | table size |
| 02h | 256 BYTEs | values used to sort characters 00h to FFh |

### 24-116 Format of filename terminator table:

| *Offset* | *Size* | *Description* | |
|---|---|---|---|
| 00h | WORD | table size (not counting this word) | |
| 02h | BYTE | *unknown* (01h for MS-DOS 3.30-6.00) | |
| 03h | BYTE | lowest permissible character value for filename | |
| 04h | BYTE | highest permissible character value for filename | |
| 05h | BYTE | *unknown* (00h for MS-DOS 3.30-6.00) | |
| 06h | BYTE | first excluded character in range | all characters in this range are illegal |
| 07h | BYTE | last excluded character in range | |
| 08h | BYTE | *unknown* (02h for MS-DOS 3.30-6.00) | |
| 09h | BYTE | number of illegal (terminator) characters | |
| 0Ah | N BYTEs | characters which terminate a filename: .”/\[ ]:\|<>+=;, | |

**Note:** Partially documented for DOS 5+, but undocumented for earlier versions.
.comment the above < > should not be changed into a not-equals

### 24-117 Format of filename uppercase table:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | table size |
| 02h | 128 BYTEs | uppercase equivalents (if any) of characters 80h to FFh |

### 24-118 Format of DBCS lead byte table:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | length |
| 02h | 2N BYTEs | start/end for N lead byte ranges |
| | WORD | 0000h (end of table) |

---

## INT 21h Functions 6520h to 6522h

*DOS 4+*

### *COUNTRY-DEPENDENT CHARACTER CAPITALIZATION*

**Registers at call:**
AH = 65h
AL = function
    20h capitalize character
        DL = character to capitalize
    21h capitalize string
        DS:DX -> string to capitalize
        CX = length of string
    22h capitalize ASCIZ string
        DS:DX -> ASCIZ string to capitalize

**Return Registers:**
CF set on error
    AX = error code (see Table 24-2)
CF clear if successful

DL = capitalized character

**Details:** These calls were undocumented prior to the release of MS-DOS 5.0.

---

## INT 21h Function 6523h

*DOS 4+ internal (undocumented)*

### *DETERMINE IF CHARACTER REPRESENTS YES/NO RESPONSE*

**Registers at call:**
AX = 6523h
DL = character

DH = second character of double-byte character (if applicable)

**Return Registers:**
CF set on error
CF clear if successful
    AX = type
        00h no
        01h yes
        02h neither yes nor no

---

### INT 21h Functions 65A0h to 65A2h
### *COUNTRY-DEPENDENT FILENAME CAPITALIZATION*

*DOS 4+ internal (undocumented)*

**Registers at call:**
AH = 65h
AL = function

**Return Registers:**
CF set on error
    AX = error code (see Table 24-2)
CF clear if successful

A0h capitalize filename character
    DL = character to capitalize
A1h capitalize counted filename string
    DS:DX -> filename string to capitalize
    CX = length of string
A2h capitalize ASCIZ filename
    DS:DX -> ASCIZ filename to capitalize

DL = capitalized character

**BUG:** These functions are nonfunctional in DOS 4.00 through 6.00 because the code sets a pointer to the appropriate capitalization table depending on the high bit of AL, but doesn't clear the bit before branching by function number.

---

### INT 21h Function 6601h
### *GET GLOBAL CODE PAGE TABLE*

*DOS 3.3+*

**Registers at call:**
AX = 6601h

**Return Registers:**
CF set on error
    AX = error code (see Table 24-2)
CF clear if successful
    BX = active code page (see Table 24-119)
    DX = system code page

**See also:** Function 6602h

---

### INT 21h Function 6602h
### *SET GLOBAL CODE PAGE TABLE*

*DOS 3.3+*

**Registers at call:**
AX = 6602h
BX = active code page (see Table 24-119)
DX = system code page (active page at boot time)

**Return Registers:**
CF set on error
    AX = error code (see Table 24-2)
CF clear if successful

**See also:** Function 6601h, DR-DOS INT 2Fh Function 14FFh (Chapter 26)

**24-119 Values for code page:**

| | |
|---|---|
| 437 | US |
| 850 | Multilingual |
| 852 | Slavic/Latin II (DOS 5+) |
| 857 | Turkish |
| 860 | Portugal |
| 861 | Iceland |
| 863 | Canada (French) |
| 865 | Norway/Denmark |

## INT 21h Function 67h

*DOS 3.3+*

### *SET HANDLE COUNT*

**Purpose:** Adjust the size of the per-process open file table, thus raising or lowering the limit on the number of files the caller can open simultaneously.

**Registers at call:**
AH = 67h
BX = size of new file handle table for process

**Return Registers:**
CF clear if successful
CF set on error
    AX = error code (see Table 24-2)

**Details:** If BX is 20 or less, no action is taken if the handle limit has not yet been increased, and the table is copied back into the PSP if the limit is currently greater than 20 handles.

For file handle tables of more than 20 handles, DOS 3.30 never reuses the same memory block, even if the limit is being reduced; this can lead to memory fragmentation as a new block is allocated and the existing one freed.

Only the first 20 handles are copied to child processes in DOS 3.3-6.0.

Increasing the file handles here will not, in general, increase the number of files that can be opened using the runtime library of a high-level language such as C.

**BUGS:** The original release of DOS 3.30 allocates a full 64K for the handle table on requests for an even number of handles.

DR-DOS 3.41 and 5.0 will lose track of any open file handles beyond the portion of the JFT retained after the call; MS-DOS will indicate error 04h if any of the JFT entries to be removed are open.

**See also:** Function 26h, European MS-DOS Function 86h (Chapter 25)

---

## INT 21h Function 68h

*DOS 3.3+*

### *"FFLUSH" - COMMIT FILE*

**Registers at call:**
AH = 68h
BX = file handle

**Return Registers:**
CF clear if successful
    all data still in DOS disk buffers is written to disk immediately, and the file's directory entry is updated
CF set on error
    AX = error code (see Table 24-2)

**See also:** Functions 5D01h and 6Ah, Network Redirector INT 2Fh Function 1107h (NI)

---

## INT 21h Function 69h

*DOS 4+ internal (undocumented)*

### *GET/SET DISK SERIAL NUMBER*

**Registers at call:**
AH = 69h
AL = subfunction
    00h get serial number
    01h set serial number
BL = drive (0=default, 1=A, 2=B, etc.)
DS:DX -> disk info (see Table 24-120)

**Return Registers:**
CF set on error
    AX = error code (see Table 24-2)
CF clear if successful
    AX destroyed
    (AL = 00h) buffer filled with appropriate values from extended BPB
    (AL = 01h) extended BPB on disk set to values from buffer

**Details:** This function does not generate a critical error; all errors are returned in AX. Error 0005h is returned if no extended BPB is present on the disk.

This function does not work on many network drives (error 0001h).

The buffer after the first two bytes is an exact copy of bytes 27h through 3Dh of the extended BPB on disk.

This function is supported under Novell NetWare versions 2.0A through 3.11; the returned serial number is the one a DIR would display, the volume label is the NetWare volume label, and the file system is set to "FAT16".

The serial number is computed from the current date and time when the disk is created; the first part is the sum of the seconds/hundredths and month/day, the second part is the sum of the hours/minutes and year.

The volume label which is read or set is the one stored in the extended BPB on disks formatted with DOS 4.0+, rather than the special root directory entry used by the DIR command in COMMAND.COM (use Function 11h to find that volume label).

**See also:** Function 440Dh

**24-120 Format of disk info:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | info level (zero) |
| 02h | DWORD | disk serial number (binary) |
| 06h | 11 BYTEs | volume label or "NO NAME " if none present |
| 11h | 8 BYTEs | (AL=00h only) filesystem type--string "FAT12 " or "FAT16 " |

---

### INT 21h Function 6Ah
### *COMMIT FILE*

*DOS 4+*
*(undocumented)*

**Registers at call:**
AH = 6Ah
BX = file handle

**Return Registers:**
CF clear if successful
    AH = 68h
CF set on error
    AX = error code (06h) (see Table 24-2)

**Details:** This call is identical to Function 68h in DOS 5.0-6.0; it is not known whether this is the case in DOS 4.x.

**See also:** Function 68h

---

### INT 21h Function 6Bh
### *IFS IOCTL*

*DOS 4.0 internal*
*(undocumented)*

**Registers at call:**
AH = 6Bh
AL = subfunction
    00h *unknown*
        DS:SI -> *Current Directory Structure*
        CL = drive (1=A:)
    01h *unknown*
        DS:SI -> *unknown*
        CL = *file handle*
    02h *unknown*
        DS:SI -> *Current Directory Structure*
        DI = *unknown*
        CX = drive (1=A:)

**Return Registers:**
CF set on error
    AX = error code (see Table 24-2)
CF clear if successful

**Details:** This call is passed through to INT 2Fh Function 112Fh (see *Network Interrupts*) with AX on top of stack.

**See also:** Function 6Bh below, Network Redirector INT 2Fh Function 112Fh (NI)

---

### INT 21h Function 6Bh
### *NULL FUNCTION*

*DOS 5+*
*(undocumented)*

**Registers at call:**
AH = 6Bh

**Return Registers:**
AL = 00h

**Details:** This function does nothing and returns immediately.

**See also:** Function 6Bh above

### INT 21h Function 6C00h

DOS 4+

### *EXTENDED OPEN/CREATE*

**Registers at call:**
AX = 6C00h
BL = open mode as in AL for normal open (see Table 24-19)
    bit 7: inheritance
    bits 4-6: sharing mode
    bit 3 reserved
    bits 0-2: access mode
BH = flags
    bit 6 = auto commit on every write (see also Function 68h)
    bit 5 = return error rather than doing INT 24h
CX = create attribute (see Table 24-21)
DL = action if file exists/does not exists
    bits 7-4 action if file does not exist
        0000 fail
        0001 create
    bits 3-0 action if file exists
        0000 fail
        0001 open
        0010 replace/open
DH = 00h (reserved)
DS:SI -> ASCIZ file name

**Return Registers:**
CF set on error
    AX = error code (see Table 24-2)
CF clear if successful
    AX = file handle
    CX = status
        1 file opened
        2 file created
        3 file replaced

**Details:** The PC LAN Program only supports DL=01h, DL=10h/sharing=compatibility, and DL=12h.

**See also:** Functions 3Ch, 3Dh, and 6C01h

---

### INT 21h Function 6Dh

DOS 5+ ROM
*(undocumented)*

### *FIND FIRST ROM PROGRAM*

**Registers at call:**
AH = 6Dh
DS:DX -> ASCIZ program name

**Return Registers:**
CF clear if found
    [DTA] = ASCIZ name found
CF set if not found

**Conflicts:** OS/2 (Chapter 26)

**See also:** Functions 6Eh and 6Fh

---

### INT 21h Function 6Eh

DOS 5+ ROM
*(undocumented)*

### *FIND NEXT ROM PROGRAM*

**Registers at call:**
AH = 6Eh
[DTA] = result of previous FindFirst ROM

**Return Registers:**
CF clear if found
    [DTA] = ASCIZ name found
CF set if not found

**Conflicts:** OS/2 (Chapter 26)

**See also:** Function 6Dh

---

## INT 21h Functions 6F00h and 6F01h

### *GET/SET ROM SCAN START ADDRESS*

*DOS 5+ ROM (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = function | |
| 6F00h get start address | BX = current ROM scan starting segment |
| 6F01h set start address | |
| BX = new ROM scan starting address | |

**Conflicts:** OS/2 (Chapter 26)
**See also:** Functions 6Dh, 6F02h, and 6F03h

---

## INT 21h Functions 6F02h and 6F03h

### *GET/SET EXCLUSION REGION LIST*

*DOS 5+ ROM (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = function | |
| 6F02h get list<br>ES:BX -> buffer for exclusion region list (see Table 24-121) | ES:BX = 0000h:0000h on error, unchanged if buffer filled |
| 6F03h set list<br>DS:DX -> new exclusion region list (see Table 24-121) | nothing |

**Details:** For Function 6F03h, DOS saves only the pointer and assumes that the contents of the list are never changed, and that regions do not overlap.
**Conflicts:** OS/2 (Chapter 26)
**See also:** Functions 6F00h and 6F01h

**24-121 Format of exclusion region list:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of entries |
| 02h | 2N WORDs | start/end segments of N excluded regions |

---

## INT 21h Function F8h

### *SET OEM INT 21h HANDLER*

*DOS v2.11*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = F8h | nothing |
| DS:DX -> OEM INT 21h handler for functions F9h to FFh<br>FFFFh:FFFFh disables OEM handler | |

**Details:** This function is supported by Toshiba T1000 ROM MS-DOS 2.11.
Calls to Functions F9h through FFh will return AL=00h if no handler has been set.
Handler is called with all registers exactly as set by caller, and should exit with IRET.
**Conflicts:** DoubleDOS (Chapter 49)
**See also:** Function F9h

---

## INT 21h Functions F9h to FFh

### *OEM FUNCTIONS*

*DOS v2.11*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = F9h to FFh | nothing unless OEM handler set |

**Conflicts:** CED and DOSED (Chapter 13), PCMANAGE/DCOMPRES (Chapter 16), DOS/4GW (Chapter 35), GO32 (Chapter 37), DoubleDOS (Chapter 49), VDEFEND/VSAFE/VWATCH (Chapter 59), FLU_SHOT+ (Chapter 59), Viruses (Chapter 59), Topware Network Operating System
**See also:** Function F8h

### INT 2Fh Function 0800h
### *INSTALLATION CHECK*

*DRIVER.SYS support*
*(undocumented)*

**Purpose:** Determine whether the internal support code used by DRIVER.SYS is present.

**Registers at call:**
AX = 0800h

**Return Registers:**
AL = status
  00h not installed, OK to install
  01h not installed, not OK to install
  FFh supported

**Details:** This function is supported by DR-DOS 5.0.

### INT 2Fh Function 0801h
### *ADD NEW BLOCK DEVICE*

*DRIVER.SYS support*
*(undocumented)*

**Registers at call:**
AX = 0801h
DS:DI -> drive data table (see Table 24-127)

**Return Registers:**
AX,BX,SI,ES destroyed

**Details:** This function moves down the internal list of drive data tables, copying and modifying the drive description flags word for tables referencing the same physical drive. The new data table is then appended to the chain of tables.

This function is supported by DR-DOS 5.0.

**See also:** Function 0803h

### INT 2Fh Function 0802h
### *EXECUTE DEVICE DRIVER REQUEST*

*DRIVER.SYS support*
*(undocumented)*

**Registers at call:**
AX = 0802h
ES:BX -> device driver request header
  (see Table 24-123)

**Return Registers:**
request header updated as per requested operation

**Details:** This function is supported by DR-DOS 5.0.

DOS 3.2 executes this function on any AL value from 02h through F7h; DOS 4+ executes this function on AL=02h and AL=04h-F7h.

The command codes and structures described below apply to all drivers which support the appropriate commands; this call is just one of a number of ways in which a device driver request may be invoked.

**See also:** Functions 0800h, 0801h, and 0803h; INT 21h Function 52h (above); European MS-DOS INT 21h Functions 99h and 9Ah (Chapter 25)

**24-122 Device driver command codes:**

| | |
|---|---|
| 00h | INIT |
| 01h | MEDIA CHECK (block devices) |
| 02h | BUILD BPB (block devices) |
| 03h | IOCTL INPUT |
| 04h | INPUT |
| 05h | NONDESTRUCTIVE INPUT, NO WAIT (character devices) |
| 06h | INPUT STATUS (character devices) |
| 07h | INPUT FLUSH (character devices) |
| 08h | OUTPUT |
| 09h | OUTPUT WITH VERIFY |
| 0Ah | OUTPUT STATUS (character devices) |
| 0Bh | OUTPUT FLUSH (character devices) |

| | |
|---|---|
| 0Ch | IOCTL OUTPUT |
| 0Dh | (DOS 3+) DEVICE OPEN |
| 0Eh | (DOS 3+) DEVICE CLOSE |
| 0Fh | (DOS 3+) REMOVABLE MEDIA (block devices) |
| 10h | (DOS 3+) OUTPUT UNTIL BUSY (character devices) |
| 11h | (European 4.0) STOP OUTPUT (console screen drivers) |
| 12h | (European 4.0) RESTART OUTPUT (console screen drivers) |
| 13h | (DOS 3.2+) GENERIC IOCTL |
| 15h | (European MS-DOS 4.0) RESET UNCERTAIN MEDIA FLAG |
| 17h | (DOS 3.2+) GET LOGICAL DEVICE |
| 18h | (DOS 3.2+) SET LOGICAL DEVICE |
| 19h | (DOS 5+) CHECK GENERIC IOCTL SUPPORT |

**24-123 Format of device driver request header:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | length of request header |
| 01h | BYTE | subunit within device driver |
| 02h | BYTE | command code (see Table 24-122) |
| 03h | WORD | status (filled in by device driver)<br>bit 15: error<br>bits 14-11: reserved<br>bit 10: *unknown*, set by DOS kernel on entry to some driver calls<br>bit 9: busy<br>bit 8: done (may be clear on return under European MS-DOS 4.0)<br>bits 7-0: error code if bit 15 set (see Table 24-124) |
| **---DOS---** | | |
| 05h | 4 BYTEs | reserved (unused in DOS 2.x and 3.x) |
| 09h | DWORD | (European MS-DOS 4.0 only) pointer to next request header in device's request queue<br>(other versions) reserved (unused in DOS 2.x and 3.x) |
| **---STARLITE (Embedded DOS)---** | | |
| 05h | DWORD | pointer to next request header |
| 09h | 4 BYTEs | reserved |
| **---command code 00h---** | | |
| 0Dh | BYTE | (return) number of units |
| 0Eh | DWORD | (call) pointer to DOS device helper function (see Table 24-125) (European MS-DOS 4.0 only)<br>(call) pointer past end of memory available to driver (DOS 5+)<br>(return) address of first free byte following driver |
| 12h | DWORD | (call) pointer to commandline arguments<br>(return) pointer to BPB array (block drivers) or 0000h:0000h (character drivers) |
| 16h | BYTE | (DOS 3+) drive number for first unit of block driver (0=A) |
| **---European MS-DOS 4.0---** | | |
| 17h | DWORD | pointer to function to save registers on stack |
| **---DOS 5+ ---** | | |
| 17h | WORD | (return) error-message flag<br>0001h MS-DOS should display error message on initialization failure |
| **---command code 01h---** | | |
| 0Dh | BYTE | media descriptor |
| 0Eh | BYTE | returned status |

| | | |
|---|---|---|
| | | 00h don't know, 01h media unchanged, FFh media has been changed |
| 0Fh | DWORD | (return, DOS 3+) pointer to previous volume ID if OPEN/CLOSE/RM bit in device header is set and disk changed |

**---command code 02h---**

| | | |
|---|---|---|
| 0Dh | BYTE | media descriptor |
| 0Eh | DWORD | transfer address<br>pointer to scratch sector if NON-IBM FORMAT bit in device header set<br>pointer to first FAT sector otherwise |
| 12h | DWORD | pointer to BPB (set by driver) (see Table 24-95) |

**---command codes 03h,0Ch--- (see also INT 21h Functions 4402h and 4403h)**

| | | |
|---|---|---|
| 0Dh | BYTE | media descriptor (block devices only) |
| 0Eh | DWORD | transfer address |
| 12h | WORD | (call) number of bytes to read/write<br>(return) actual number of bytes read or written |

**---command codes 04h,08h,09h---**

| | | |
|---|---|---|
| 0Dh | BYTE | media descriptor (block devices only) |
| 0Eh | DWORD | transfer address |
| 12h | WORD | byte count (character devices) or sector count (block devices) |
| 14h | WORD | starting sector number (block devices only) |
| 16h | DWORD | (DOS 3+) pointer to volume ID if error 0Fh returned |
| 1Ah | DWORD | (DOS 4+) 32-bit starting sector number (block devices with device attribute word bit 1 set only) if starting sector number above is FFFFh (see Table 24-84) |

**---command code 05h---**

| | | |
|---|---|---|
| 0Dh | BYTE | byte read from device if BUSY bit clear on return |

**---command codes 06h,07h,0Ah,0Bh,0Dh,0Eh,0Fh---**

no further fields

**---command code 10h---**

| | | |
|---|---|---|
| 0Dh | BYTE | unused |
| 0Eh | DWORD | transfer address |
| 12h | WORD | (call) number of bytes to write<br>(return) actual number of bytes written |

**---command codes 11h,12h---**

| | | |
|---|---|---|
| 0Dh | BYTE | reserved |

**---command code 15h---**

no further fields

**---command codes 13h,19h---**

| | | |
|---|---|---|
| 0Dh | BYTE | category code<br>00h unknown<br>01h COMn:<br>03h CON<br>05h LPTn:<br>07h mouse (European MS-DOS 4.0)<br>08h disk<br>9Eh (STARLITE) Media Access Control driver |
| 0Eh | BYTE | function code<br>00h (STARLITE) MAC Bind request |
| 0Fh | WORD | copy of DS at time of IOCTL call (apparently unused in DOS 3.3)<br>SI contents (European MS-DOS 4.0) |

## INT 2Fh Function 0802h

| | | |
|---|---|---|
| 11h | WORD | offset of device driver header<br>DI contents (European MS-DOS 4.0) |
| 13h | DWORD | pointer to parameter block from INT 21h Functions 440Ch or 440Dh |

**24-124 Values for error code:**

| | |
|---|---|
| 00h | write-protect violation |
| 01h | unknown unit |
| 02h | drive not ready |
| 03h | unknown command |
| 04h | CRC error |
| 05h | bad drive request structure length |
| 06h | seek error |
| 07h | unknown media |
| 08h | sector not found |
| 09h | printer out of paper |
| 0Ah | write fault |
| 0Bh | read fault |
| 0Ch | general failure |
| 0Dh | reserved |
| 0Eh | (CD-ROM) media unavailable |
| 0Fh | invalid disk change |

**24-125 European MS-DOS 4.0 device helper function calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| DL = function | |
| 00h "SchedClock" called on each timer tick | |
| AL = tick interval in milliseconds | |
| 01h "DevDone" device I/O complete | |
| ES:BX -> request header | |
| **Note:** must update status | ZF clear if pending request |
| word first; may be called | ES:BX -> request header |
| from an interrupt handler | ZF set if no more requests |
| 02h "PullRequest" pull next request from queue | |
| DS:SI -> DWORD pointer to start of device's request queue | |

**Details:** The DWORD pointing at the request queue must be allocated by the driver and initialized to 0000h:0000h. It always points at the next request to be executed.

**Registers at call:**

DL = function
03h "PullParticular" remove specific request from queue
DS:SI -> DWORD pointer to start of device's request queue
ES:BX -> request header
04h "PushRequest" push the request onto the queue
DS:SI -> DWORD pointer to start of device's request queue
ES:BX -> request header
interrupts disabled

**Return Registers:**

05h “ConsInputFilter” keyboard input check

ZF set if request header not found

AX = character (high byte 00h if PC ASCII character)

**Details:** Called by keyboard interrupt handler so DOS can scan for special input characters

ZF set if character should be discarded
ZF clear if character should be handled normally

**Registers at call:**
DL = function

06h “SortRequest” push request in sorted order by starting sector
DS:SI -> DWORD pointer to start of device’s request queue
ES:BX -> request header
interrupts disabled

07h “SigEvent” send signal on keyboard event
AH = event identifier

09h “ProcBlock” block on event
AX:BX = event identifier (typically a pointer)
CX = timeout in ms or 0000h for never
DH = interruptable flag (nonzero if pause may be interrupted)
interrupts disabled
**Details:** Block process and schedules another to run

**Return Registers:**

AL,FLAGS destroyed

after corresponding ProcRun call
CF clear if event wakeup, set if unusual wakeup
ZF set if timeout wakeup, clear if interrupted
AL = wakeup code, nonzero if unusual wakeup
interrupts enabled
BX,CX,DX destroyed

**Registers at call:**
DL = function

0Ah “ProcRun” unblock process
AX:BX = event identifier (typically a pointer).

0Bh “QueueInit” initialize/clear character queue
DS:BX -> character queue structure (see Table 24-126)
**Details:** The queue size field

must be set before calling

**Return Registers:**

AX = number of processes awakened
ZF set if no processes awakened
BX,CX,DX destroyed

0Dh "QueueWrite" put a character in the queue
DS:BX -> character queue (see Table 24-126)
AL = character to append to end of queue.

ZF set if queue is full
ZF clear if character stored

**Registers at call:**
DL = function
0Eh "QueueRead" get a character from the queue
DS:BX -> character queue (see Table 24-126)

**Return Registers:**

ZF set if queue is empty
ZF clear if characters in queue
AL = first character in queue

BX,CX destroyed

10h "GetDOSVar" return pointer to DOS variable
AL = index of variable
03h current process ID
BX = index into variable if AL specifies an array
CX = expected length of variable
**Details:** The variables may not be modified

CF clear if successful
DX:AX -> variable
CF set on error
AX,DX destroyed

FLAGS destroyed

14h "Yield" yield CPU if higher-priority task ready to run

**Registers at call:**
DL = function
1Bh "CritEnter" begin system critical section
DS:BX -> semaphore (6 BYTEs, initialized to zero)
1Ch "CritLeave" end system critical section
DS:BX -> semaphore (6 BYTEs, initialized to zero)
**Details:** Must be called in the context of the process which called CritEnter on the semaphore

**Return Registers:**

AX,BX,CX,DX destroyed

AX,BX,CX,DX destroyed

**24-126 Format of European MS-DOS 4.0 character queue:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | size of queue in bytes |
| 02h | WORD | index of next character out |
| 04h | WORD | count of characters in the queue |
| 06h | N BYTEs | queue buffer |

## INT 2Fh Function 0803h
## *GET DRIVE DATA TABLE LIST*

*DOS 4+ DRIVER.SYS support (undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = 0803h | DS:DI -> first drive data table in list (see Table 24-127) |

**Details:** This function is not available under DR-DOS 5.0.

Due to space constraints, the drive data table formats for DOS 3.30 and Compaq DOS 3.31 have been omitted; they may be found in *Undocumented DOS.*

**See also:** Function 0801h

### 24-127 Format of DOS 4.0-6.0 drive data table:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to next table (offset FFFFh if last table) |
| 04h | BYTE | physical unit number (for INT 13h) |
| 05h | BYTE | logical drive number (0=A:) |
| 06h | 25 BYTEs | BIOS Parameter Block (see Table 24-95) |
| 1Fh | BYTE | flags<br>bit 6: 16-bit FAT instead of 12-bit<br>bit 7: unsupportable disk (all accesses will return Not Ready) |
| 20h | WORD | device-open count |
| 22h | BYTE | device type (see Table 24-34) |
| 23h | WORD | bit flags describing drive (see Table 24-128) |
| 25h | WORD | number of cylinders |
| 27h | 25 BYTEs | BIOS Parameter Block for default (highest) capacity supported |
| 40h | 6 BYTEs | reserved (part of BPB above) |
| 46h | BYTE | last track accessed |
| **---removable media---** | | |
| 47h | DWORD | time of last access in clock ticks (FFFFFFFFh if never) |
| **---fixed media---** | | |
| 47h | WORD | partition (FFFFh = primary, 0001h = extended)<br>always 0001h for DOS 5+ |
| 49h | WORD | absolute cylinder number of partition's start on physical drive (FFFFh if primary partition in DOS 4.x) |
| ------ | | |
| 4Bh | 11 BYTEs | volume label or "NO NAME " if none (apparently taken from extended boot record rather than root directory) |
| 56h | BYTE | terminating null for volume label |
| 57h | DWORD | serial number |
| 5Bh | 8 BYTEs | filesystem type ("FAT12 " or "FAT16") |
| 63h | BYTE | terminating null for filesystem type |

### 24-128 Bitfields for flags describing drive:

| | |
|---|---|
| bit 0 | fixed media |
| bit 1 | door lock ("changeline") supported |
| bit 2 | current BPB locked |
| bit 3 | all sectors in a track are the same size |
| bit 4 | physical drive has multiple logical units |
| bit 5 | current logical drive for physical drive |
| bit 6 | disk change detected |
| bit 7 | set DASD before formatting |
| bit 8 | disk reformatted |
| bit 9 | unformatted (fixed media only, disables reads and writes) |

## INT 2Fh Function 1200h
## *INSTALLATION CHECK*

*DOS 3+ internal (undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1200h | AL = FFh (for compatibility with other INT 2Fh functions) |

**24-129 DOS Utility Functions**

| | |
|---|---|
| 1200h | installation check |
| 1201h | close current file |
| 1202h | get interrupt address |
| 1203h | get DOS data segment |
| 1204h | normalize path separator |
| 1205h | output character to standard output (internal) |
| 1206h | invoke critical error (internal) |
| 1207h | make disk buffer most-recently used (internal) |
| 1208h | decrement SFT reference count |
| 1209h | flush and free disk buffer (internal) |
| 120Ah | perform critical error interrupt (internal) |
| 120Bh | signal sharing violation to user (internal) |
| 120Ch | open device and set SFT owner/mode (internal) |
| 120Dh | get date and time (internal) |
| 120Eh | mark all disk buffers unreferenced (internal) |
| 120Fh | make buffer most recently used (internal) |
| 1210h | find unreferenced disk buffer |
| 1211h | normalize ASCIZ filename |
| 1212h | get length of ASCIZ string |
| 1213h | uppercase character |
| 1214h | compare far pointers |
| 1215h | flush disk buffer (internal) |
| 1216h | get address of System File Table entry |
| 1217h | get current directory structure for drive (internal) |
| 1218h | get caller's registers (internal) |
| 1219h | *set drive* (internal) |
| 121Ah | get file's drive |
| 121Bh | set year and length of February |
| 121Ch | checksum memory (internal) |
| 121Dh | sum memory |
| 121Eh | compare filenames |
| 121Fh | build current directory structure (internal) |
| 1220h | get Job File Table entry |
| 1221h | canonicalize filename (internal) |
| 1222h | set extended error information (internal) |
| 1223h | check if character device (internal) |
| 1224h | sharing retry delay (internal) |
| 1225h | get length of ASCIZ string |
| 1226h | open file (internal) |
| 1227h | close file (internal) |
| 1228h | move file pointer (internal) |
| 1229h | read from file (internal) |
| 122Ah | set FASTOPEN entry point |
| 122Bh | IOCTL (internal) |

| | |
|---|---|
| 122Ch | get device chain |
| 122Dh | get extended error code |
| 122Eh | get or set error table addresses |
| 122Fh | set DOS version number to return |

**Note:** Functions marked (internal) are described in detail in *Network Interrupts*, since they may only be called during a DOS function call, such as from a network redirector.

---

### INT 2Fh Function 1201h
### *CLOSE CURRENT FILE*

*DOS 3+ internal (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1201h | CF set on error |
| SS = DOS DS (must be using a DOS internal stack) | BX = *unknown* |
| SDA current SFT pointer -> SFT of file to close | CX new reference count of SFT |
| | ES:DI -> SFT for file |

**See also:** INT 21h Function 3Eh (above), Function 1227h (NI), Network Redirector Function 1106h (NI)

---

### INT 2Fh Function 1202h
### *GET INTERRUPT ADDRESS*

*DOS 3+ internal (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1202h | ES:BX -> interrupt vector |
| STACK: WORD vector number | STACK unchanged |

---

### INT 2Fh Function 1203h
### *GET DOS DATA SEGMENT*

*DOS 3+ internal (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1203h | DS = data segment of IBMDOS.COM/MS-DOS.SYS |

**Details:** For DOS prior to version 5.0, the data segment is the same as the code segment.

---

### INT 2Fh Function 1204h
### *NORMALIZE PATH SEPARATOR*

*DOS 3+ internal (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1204h | AL = normalized character (forward slash turned to backslash, all others unchanged) |
| STACK: WORD character to normalize | ZF set if path separator |
| | STACK unchanged |

---

### INT 2Fh Function 1208h
### *DECREMENT SFT REFERENCE COUNT*

*DOS 3+ internal (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1208h | AX = original value of reference count |
| ES:DI -> SFT | |

**Details:** If the reference count was 1, it is set to FFFFh (since 0 indicates that the SFT is not in use). It is the caller's responsibility to set the reference count to zero after cleaning up.

This function is used by network redirectors such as MSCDEX.

**See also:** Function 1106h

---

### INT 2Fh Function 1210h
### *FIND UNREFERENCED DISK BUFFER*

*DOS 3+ internal (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1210h | ZF clear if found |
| DS:DI -> first disk buffer to check | |

**INT 2Fh Function 1201h**

DS:DI -> first unreferenced disk buffer
ZF set if not found

**Details:** In DOS 5+, this has become essentially a NOP, invoking the same code used by Function 1224h (SHARING DELAY).

**See also:** Function 120Eh

---

### INT 2Fh Function 1211h
### *NORMALIZE ASCIZ FILENAME*

*DOS 3+ internal (undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1211h | destination buffer filled with uppercase filename, with slashes turned to backslashes |
| DS:SI -> ASCIZ filename to normalize | |
| ES:DI -> buffer for normalized filename | |

**See also:** Function 121Eh, Function 1221h (NI)

---

### INT 2Fh Function 1212h
### *GET LENGTH OF ASCIZ STRING*

*DOS 3+ internal (undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1212h | CX = length of string |
| ES:DI -> ASCIZ string | |

**See also:** Function 1225h

---

### INT 2Fh Function 1213h
### *UPPERCASE CHARACTER*

*DOS 3+ internal (undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1213h | AL = uppercase character |
| STACK: WORD character to convert to uppercase | STACK unchanged |

---

### INT 2Fh Function 1214h
### *COMPARE FAR POINTERS*

*DOS 3+ internal (undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1214h | ZF set if pointers are equal, ZF clear if not equal |
| DS:SI = first pointer | |
| ES:DI = second pointer | |

---

### INT 2Fh Function 1216h
### *GET ADDRESS OF SYSTEM FILE TABLE ENTRY*

*DOS 3+ internal (undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1216h | CF clear if successful |
| BX = system file table entry number | ES:DI -> system file table entry |
| | CF set if BX greater than FILES= |

**Details:** This function is supported by DR-DOS 5+.

**See also:** Function 1220h

---

### INT 2Fh Function 121Ah
### *GET FILE'S DRIVE*

*DOS 3+ internal (undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = 121Ah | AL = drive (0 = default, 1 = A:, etc., FFh = invalid) |
| DS:SI -> filename | DS:SI -> filename without leading X: (if present) |

**See also:** INT 21h Functions 19h and 60h (above)

### INT 2Fh Function 121Bh
### *SET YEAR/LENGTH OF FEBRUARY*

*DOS 3+ internal*
*(undocumented)*

**Registers at call:**
AX = 121Bh
CL = year - 1980
DS = DOS code segment

**Return Registers:**
AL = number of days in February

**See also:** INT 21h Function 2Bh (above)

### INT 2Fh Function 121Dh
### *SUM MEMORY*

*DOS 3+ internal*
*(undocumented)*

**Registers at call:**
AX = 121Dh
DS:SI -> memory to add up
CX = 0000h
DX = limit

**Return Registers:**
AL = byte which exceeded limit
CX = number of bytes before limit exceeded
DX = remainder after adding first CX bytes
DS:SI -> byte beyond the one which exceeded the limit

**Details:** This function is used by DOS to determine year or month given day count since 1/1/80. It is supported by DR-DOS 5.0+.
**See also:** Function 121Ch

### INT 2Fh Function 121Eh
### *COMPARE FILENAMES*

*DOS 3+ internal*
*(undocumented)*

**Registers at call:**
AX = 121Eh
DS:SI -> first ASCIZ filename
ES:DI -> second ASCIZ filename

**Return Registers:**
ZF set if filenames equivalent
ZF clear if they differ

**Details:** This function is supported by DR-DOS 5.0+.
**See also:** Function 1211h, Function 1221h (NI)

### INT 2Fh Function 1220h
### *GET JOB FILE TABLE ENTRY*

*DOS 3+ internal*
*(undocumented)*

**Registers at call:**
AX = 1220h
BX = file handle

**Return Registers:**
CF set on error
    AL = 06h (invalid file handle)
CF clear if successful
    ES:DI -> JFT entry for file handle in current process

**Details:** The byte pointed at by ES:DI contains the number of the SFT for the file handle, or FFh if the handle is not open.

This function is supported by DR-DOS 5.0+.
**See also:** Function 1216h, Function 1229h (NI)

### INT 2Fh Function 1225h
### *GET LENGTH OF ASCIZ STRING*

*DOS 3+ internal*
*(undocumented)*

**Registers at call:**
AX = 1225h
DS:SI -> ASCIZ string

**Return Registers:**
CX = length of string

**Details:** This function is supported by DR-DOS 5.0+.
**See also:** Function 1212h

### INT 2Fh Function 122Ah
### *SET FASTOPEN ENTRY POINT*

*DOS 3.3+ internal*
*(undocumented)*

**Registers at call:**
AX = 122Ah

BX = entry point to set (0001h or 0002h)
DS:SI -> FASTOPEN entry point (entry point not set if SI = FFFFh for DOS 4+)

**Return Registers:**
CF set if specified entry point already set

**Details:** The entry point number in BX is ignored under DOS 3.30. Both entry points are set to same handler by DOS 4.01. DOS 5.0 and 6.0 only set entry point 1.

### 24-130 DOS 3.30+ FASTOPEN is calling convention:

**Registers at call:**
AL = 01h Lookup
    CX = *seems to be offset*
    DI = *seems to be offset*
    SI = offset in DOS DS of filename
AL = 02h insert file into FASTOPEN cache
AL = 03h delete file from FASTOPEN cache
    SI = offset in DOS DS of filename
AL = 04h purge FASTOPEN cache
    AH = subfunction (00h,01h,02h)
    ES:DI -> *unknown*
    CX = *unknown* (subfunctions 01h and 02h only)
**Note:** function 03h calls function 01h first

**Return Registers:**
CF set on error or not installed

### 24-131 PC-DOS 4.01 FASTOPEN calling convention (additional calls):

**Registers at call:**
AL = 04h *unknown*
    AH = 03h
    *others, if any, unknown*
AL = 05h *unknown*
AL = 0Bh *unknown*
AL = 0Ch *unknownN@*
*AL = 0Dh unknown*
AL = 0Eh *unknown*
AL = 0Fh *unknown*
AL = 10h *unknown*

**Return Registers:**
*unknown*

### 24-132 MS-DOS 5.0-6.0 FASTOPEN calling convention (additional calls):

**Registers at call:**
AL = 04h purge FASTOPEN cache
    AH = 03h
    *others, if any, unknown*
AL = 05h *unknown*
    DL = drive (00h = A:)
    *others, if any, unknown*
AL = 06h *unknown*
    *others, if any, unknown*

**Return Registers:**
*unknown*

## INT 2Fh Function 122Ch
## *GET DEVICE CHAIN*

*DOS 3.3+ internal (undocumented)*

**Registers at call:**
AX = 122Ch

**Return Registers:**
BX:AX -> header of second device driver (NUL is first) in driver chain

**Details:** Although this function exists in DR-DOS 5.0, it returns 0000h:0000h.
**See also:** INT 21h Function 52h

## INT 2Fh Function 122Dh

*DOS 3.3+ internal (undocumented)*

### *GET EXTENDED ERROR CODE*

| Registers at call: | Return Registers: |
|---|---|
| AX = 122Dh | AX = current extended error code |

**See also:** Function 1222h, INT 21h Function 59h (above)

---

## INT 2Fh Function 122Eh

*DOS 4+ internal (undocumented)*

### *GET OR SET ERROR TABLE ADDRESSES*

| Registers at call: | Return Registers: |
|---|---|
| AX = 122Eh | |
| DL = subfunction | |
| 00h get standard DOS error table | ES:DI -> error table |
| | (DOS 4: errors 00h-12h,50h-5Bh) |
| | (DOS 5: errors 00h-26h,4Fh,51h-59h) |
| 01h set standard DOS error table | |
| ES:DI -> error table | |
| 02h get parameter error table (errors 00h-0Ah) | ES:DI -> error table |
| 03h set parameter error table | |
| ES:DI -> error table | |

| Registers at call: | Return Registers: |
|---|---|
| AX = 122Eh | |
| DL = subfunction | |
| 04h get critical/SHARE error table (errors 13h-2Bh) | ES:DI -> error table |
| 05h set critical/SHARE error table | |
| ES:DI -> error table | ES:DI -> error table or 0000h:0000h |
| 06h get *unknown* error table | |
| 07h set *unknown* error table | |
| ES:DI -> error table | ES:DI -> FAR procedure to fetch error message |
| 08h get error message retriever (see Table 24-134) | |
| 09h set *unknown* error table | |
| ES:DI -> error table | |

**Details:** If the returned segment on a "get" is 0001h, then the offset specifies the offset of the error message table within COMMAND.COM, and the procedure returned by DL=08h should be called.

DOS 5+ COMMAND.COM does not allow setting any of the addresses (calls with DL odd are ignored); they are always returned with segment 0001h. For DOS 5.0, the standard and critical/SHARE error tables are combined into a single error table.

**See also:** INT 21h Function 59h (above), Function 0500h (Chapter 27)

**24-133 Format of DOS 4.x error table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | FFh |
| 01h | 2 BYTEs | 04h,00h (*DOS version*) |
| 03h | BYTE | number of error headers following |
| 04h | 2N WORDs | table of all error headers for table |

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | error message number |
| 02h | WORD | offset of error message from start of header |
| | | error messages are count byte followed by msg |

**Note:** DOS 5 error tables consist of one word per error number; each word contains either the offset of a counted string or 0000h.

### 24-134 Error retrieval function calling convention:

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = error number | ES:DI -> error message (counted string) |
| DI = offset of error table | |

**Details:** This function needs to access COMMAND.COM if the messages were not loaded into memory permanently with /MSG; the caller should assume that the returned message will be overwritten by the next call of the function.

This function is supported by DR-DOS 5.0.

### 24-135 Values for parameter errors:

| | |
|---|---|
| 01h | Too many parameters |
| 02h | Required Parameter missing |
| 03h | Invalid switch |
| 04h | Invalid keyword |
| 06h | Parameter value not in allowed range |
| 07h | Parameter value not allowed |
| 08h | Parameter value not allowed |
| 09h | Parameter format not correct |
| 0Ah | Invalid parameter |
| 0Bh | Invalid parameter combination |

## INT 2Fh Function 122Fh
## *SET DOS VERSION NUMBER TO RETURN*

*DOS 4.x internal (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 122Fh | nothing |
| DX = DOS version number (0000h = return true DOS version) | |

**Details:** This function is not available under DR-DOS 5.0 or 6.0.

**See also:** INT 21h Functions 30h and 3306h (above)

## INT 2Fh Function 13h
## *SET DISK INTERRUPT HANDLER*

*DOS 3.2+ (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 13h | DS:DX and ES:BX set to values from previous invocation of this function, or original handlers set by MS-DOS if this is the first call |
| DS:DX -> interrupt handler disk driver calls on read/write | |
| ES:BX = address to restore INT 13h to on system halt (exit from root shell) or warm boot (INT 19h) | |

**Details:** IO.SYS hooks INT 13h and inserts one or more filters ahead of the original INT 13h handler. The first is for disk change detection on floppy drives, the second is for tracking formatting calls and correcting DMA boundary errors, the third is for working around problems in a particular version of IBM's ROM BIOS.

Before the first call, ES:BX points at the original BIOS INT 13h; DS:DX also points there unless IO.SYS has installed a special filter for hard disk reads (on systems with model byte FCh and BIOS dated "01/10/84" only), in which case it points at the special filter.

Most DOS 3.3+ disk access is via the vector in DS:DX, although a few functions are still invoked via an INT 13h instruction.

This is a dangerous security loophole for any virus-monitoring software which does not trap this call ("INT13", "Nomenklatura", and many Bulgarian viruses are known to use it to get the original ROM entry point).

**Conflicts:** MS-NET (NI)
**See also:** INT 13h Function 01h, INT 19h,VIRUS INT 9Dh

---

### INT 2Fh Functions 4601h and 4602h
### *UNDETERMINED FUNCTION*

*DOS 5+ kernel (undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = 4601h and 4602h | *unknown* |

**Details:** Function 4601h copies the MCB following the caller's PSP memory block into the DOS data segment; Function 4602h copies the previously copied MCB from the DOS data segment into the MCB following the caller's PSP memory block.
**Conflicts:** Windows/286 DOS Extender (Chapter 37)

---

### INT 2Fh Function 4A00h, Subfunction 0000h
### *FLOPPY-DISK LOGICAL DRIVE CHANGE NOTIFICATION*

*DOS 5+ Callout (undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = 4A00h<br>CX = 0000h<br>DH = new drive number<br>DL = current drive number | CX = FFFFh to skip "Insert diskette for drive X:" message |

**Details:** This function is called by MS-DOS 5.0+ IO.SYS just before displaying the message "Insert diskette for drive X:" on single-floppy systems

---

### INT 2Fh Function 4A01h
### *QUERY FREE HMA SPACE*

*DOS 5+*

| Registers at call: | Return Registers: |
|---|---|
| AX = 4A01h | BX = number of bytes available in HMA (0000h if DOS not using HMA)<br>ES:DI -> start of available HMA area (FFFFh:FFFFh if not using HMA) |

**Details:** This function is called by Windows 3.1 DOSX.EXE.
**See also:** Function 4A02h, XMS Function 4310h (Chapter 38)

---

### INT 2Fh Function 4A02h
### *ALLOCATE HMA SPACE*

*DOS 5+*

| Registers at call: | Return Registers: |
|---|---|
| AX = 4A02h<br>BX = number of bytes | ES:DI -> start of allocated HMA block or FFFFh:FFFFh<br>BX = number of bytes actually allocated (rounded up to next paragraph for DOS 5.0 and 6.0) |

**Details:** This function is not valid unless DOS is loaded in the HMA (DOS=HIGH). It is called by Windows 3.1 DOSX.EXE.
**See also:** Function 4A01h

## Other Services

### INT 15h Function 4900h
### *GET DOS TYPE*

*Far East DOS*

**Registers at call:**
AX = 4900h

**Return Registers:**
CF clear if successful
- AH = 00h
- BL = type of DOS running
  - 00h DOS/V
  - 01h DOS/J or DOS/K (early IBM Japan versions of MS-DOS)

CF set on error
AH = 86h (function not supported)

**Details:** In practice, DOS/J returns AH=86h; AX DOS does not support this call.

**See also:** INT 21h Function 30h (above)

---

### INT 22h
*DOS 1+*

### *PROGRAM TERMINATION ADDRESS*

**Purpose:** This vector specifies the address of the routine which is to be given control after a program is terminated; it should never be called directly, since it does not point at an interrupt handler.

**Details:** This vector is restored from the DWORD at offset 0Ah in the PSP during termination, and then a FAR JMP is performed to the address in INT 22h. It normally points at the instruction immediately following the INT 21h Function 4Bh call which loaded the current program.

**Conflicts:** Comtrol Hostess i/ISA Debugger (Chapter 57)

**See also:** INT 20h; INT 21h Functions 00h, 31h, and 4Ch (above)

---

### INT 23h
*DOS 1+*
*Callout*

### *CONTROL-C/CONTROL-BREAK HANDLER*

**Purpose:** DOS invokes this interrupt whenever it detects a Control-C or Control-Break keystroke; it should never be called directly. MS-DOS 1.25 also invokes INT 23h on a divide overflow (INT 00h).

**Registers at call (DOS 1.x):**
none

**Return Registers:**
AH = 00h abort program
if all registers preserved, restart DOS call

**Register at call (DOS 2+):**
CF clear

**Return Registers:**
all registers preserved
return via RETF or RETF 2 with CF set
    DOS will abort program with errorlevel 0
else (RETF/RETF 2 with CF clear or IRET)
    interrupted DOS call is restarted

DOS 2+ remembers the stack pointer before calling INT 23h, and if it is not the same on return, pops and discards the top word; this is what permits a return with RETF as well as IRET or RETF 2.

Any DOS call may safely be made within the INT 23h handler, although the handler must to check for a recursive invocation if it does call DOS.

**Conflicts:** Comtrol Hostess i/ISA Debugger (Chapter 57)

**See also:** INT 1Bh (Chapter 4)

---

### INT 24h
*DOS 1+*
*Callout*

### *CRITICAL ERROR HANDLER*

**Purpose:** DOS invokes this interrupt when a critical (usually hardware) error is encountered; it should never be called directly.

**Registers at call:**
AH = type and processing flags
  (see Table 24-136)
AL = drive number if AH bit 7 clear
BP:SI -> device driver header
  (BP:[SI+4] bit 15 set if character device)
DI low byte contains error code if AH bit 7 set (see Table 24-137)
STACK: (see Table 24-138)

**Return Registers:**
AL = action code
    00h ignore error and continue processing request
    01h retry operation
    02h terminate program through the equivalent of INT 21h Function 4Ch (INT 20h for DOS 1.x)
    03h fail system call in progress
SS,SP,DS,ES,BX,CX,DX preserved

**See also:** European MS-DOS INT 21h Function 95h (Chapter 25)
**Details:** The only DOS calls the handler may make are INT 21h Function 01h-0Ch, 30h, and 59h.

If the handler returns to the application by popping the stack, DOS will be in an unstable state until the first call with AH > 0Ch.

For DOS 3.1+, IGNORE (AL=00h) is turned into FAIL (AL=03h) on network critical errors.

If IGNORE is specified but not allowed, it is turned into FAIL. If RETRY is specified but not allowed, it is turned into FAIL. If FAIL is specified but not allowed, it is turned into ABORT.

(DOS 3+) If a critical error occurs inside the critical error handler, the DOS call is automatically failed.

**24-136 Bitfields for error type and processing flags:**

| | | |
|---|---|---|
| bit 7 | clear | disk I/O error |
| | set | bad FAT image in memory (block device)<br>error code in DI (character device) |
| bit 6 | unused | |
| bit 5 | Ignore allowed (DOS 3+) | |
| bit 4 | Retry allowed (DOS 3+) | |
| bit 3 | Fail allowed (DOS 3+) | |
| bits 2,1 | disk area of error (00 DOS area, 01 FAT, 10 root dir, 11 data area) | |
| bit 0 | set if write, clear if read | |

**24-137 Error codes:**

| | |
|---|---|
| 00h | write-protection violation attempted |
| 01h | unknown unit for driver |
| 02h | drive not ready |
| 03h | unknown command given to driver |
| 04h | data error (bad CRC) |
| 05h | bad device driver request structure length |
| 06h | seek error |
| 07h | unknown media type |
| 08h | sector not found |
| 09h | printer out of paper |
| 0Ah | write fault |
| 0Bh | read fault |
| 0Ch | general failure |
| 0Dh | (DOS 3+) sharing violation |
| 0Eh | (DOS 3+) lock violation |
| 0Fh | invalid disk change |
| 10h | (DOS 3+) FCB unavailable |
| 11h | (DOS 3+) sharing buffer overflow |
| 12h | (DOS 4+) code page mismatch |
| 13h | (DOS 4+) out of input |
| 14h | (DOS 4+) insufficient disk space |

**24-138 INT 24h Stack Frame:**

| | | |
|---|---|---|
| STACK: | DWORD | return address for INT 24h call |
| | WORD | flags pushed by INT 24h |
| | WORD | original AX on entry to INT 21h |
| | 8 WORDs | BX,CX,DX,SI,DI,BP,DS,ES |
| | DWORD | return address for INT 21h call |
| | WORD | flags pushed by INT 21h |

## INT 25h

*DOS 1+*

## *ABSOLUTE DISK READ*

**Registers at call:**

AL = drive number (00h = A:, 01h = B:, etc.)

**---partitions <= 32M---**

CX = number of sectors to read

DX = starting logical sector number (0000h - highest sector on drive)

DS:BX -> buffer for data

**---partitions > 32M---**

CX = FFFFh

DS:BX -> disk read packet (see Table 24-139)

**Return Registers:**

CF clear if successful

CF set on error

AH = status (see Table 24-140)

AL = error code (see Table 24-137)

AX = 0207h if more than 64K sectors on drive

may destroy all other registers except segment registers

**Details:** The original flags are left on the stack, and must be popped by the caller.

This call bypasses the DOS filesystem.

The partition is potentially greater than 32M (and requires the second form of the call) if bit 1 of the device attribute word in the device driver is set (see Table 24-84).

On most block devices, reads will be truncated if they extend past the end of the specified segment.

**BUGS:** DOS 3.1 through 3.3 set the word at ES:[BP+1Eh] to FFFFh if AL is an invalid drive number.

DR-DOS 3.41 will return with a jump instead of RETF, leaving the wrong number of bytes on the stack; use the huge-partition version for all partition sizes under DR-DOS 3.41.

**Conflicts:** Stacker (Chapter 16), PC-CACHE.SYS (Chapter 17)

**See also:** INT 26h, BIOS INT 13h Function 02h (Chapter 10)

**24-139 Format of disk read/write packet:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | logical sector number |
| 04h | WORD | number of sectors to transfer |
| 06h | DWORD | transfer address |

**24-140 Values for status:**

| | |
|---|---|
| 80h | device failed to respond (timeout) |
| 40h | seek operation failed |
| 20h | controller failed |
| 10h | data error (bad CRC) |
| 08h | DMA failure |
| 04h | requested sector not found |
| 03h | write-protected disk (INT 26h only) |
| 02h | bad address mark |
| 01h | bad command |

## INT 26h

*DOS 1+*

## *ABSOLUTE DISK WRITE*

**Registers at call:**

AL = drive number (00h = A:, 01h = B:, etc.)

**---partitions <= 32M---**

CX = number of sectors to write

DX = starting logical sector number

(0000h - highest sector on drive)
DS:BX -> data to write
**---partitions > 32M---**
CX = FFFFh
DS:BX -> disk write packet
(see Table 24-139)

**Return Registers:**
CF clear if successful
CF set on error
AH = status (see Table 24-140)
AL = error code (see Table 24-137)
AX = 0207h if more than 64K sectors on drive
may destroy all other registers except segment registers

**Details:** The original flags are left on the stack, and must be popped by the caller.

This call bypasses the DOS filesystem, though DOS 5+ invalidates any disk buffers referencing sectors which are written with this call.

The partition is potentially >32M (and requires the second form of the call) if bit 1 of device attribute word in device driver is set (see Table 24-84).

**BUGS:** DOS 3.1 through 3.3 set the word at ES:[BP+1Eh] to FFFFh if AL is an invalid drive number.

DR-DOS 3.41 will return with a jump instead of RETF, leaving the wrong number of bytes on the stack; use the huge-partition version for all partition sizes under DR-DOS 3.41.

**Conflicts:** Comtrol Hostess i/ISA Debugger (Chapter 57)

**See also:** INT 25h, BIOS INT 13h Function 03h (Chapter 10)

---

## INT 27h
*DOS 1+*

### *TERMINATE AND STAY RESIDENT*

**Registers at call:**
DX = number of bytes to keep resident (max FFF0h)
CS = segment of PSP

**Return Registers:**
never returns

**Details:** This is an obsolete call; use INT 21h Function 31h instead.

INT 22h, INT 23h, and INT 24h are restored from the PSP.

This call does not close any open files.

The minimum number of bytes which will remain resident is 110h for DOS 2.x and 60h for DOS 3+; there is no minimum for DOS 1.x, which implements this service in COMMAND.COM rather than the DOS kernel.

**Conflicts:** Comtrol Hostess i/ISA Debugger (Chapter 57)

**See also:** INT 21h Function 31h (above)

---

## INT 28h
*DOS 2+*
*Callout*

### *DOS IDLE INTERRUPT*

**Purpose:** This interrupt is invoked each time one of the DOS character input functions loops while waiting for input. Since a DOS call is in progress even though DOS is actually idle during such input waits, hooking this function is necessary to allow a TSR to perform DOS calls while the foreground program is waiting for user input. The INT 28h handler may invoke any INT 21h function except functions 00h through 0Ch.

**Registers at call:**
SS:SP = top of MS-DOS stack for I/O functions

**Return Registers:**
all registers preserved

**Details:** Under DOS 2.x, the critical error flag (the byte immediately after the InDOS flag) must be set in order to call DOS functions 50h/51h from the INT 28h handler without destroying the DOS stacks.

Calls to INT 21h Function 3Fh,40h from within an INT 28h handler may not use a handle which refers to CON.

At the time of the call, the InDOS flag (see INT 21h Function 34h) is normally set to 01h; if larger, DOS is truly busy and should not be reentered.

The default handler is an IRET instruction.

Supported in OS/2 compatibility box.

The *MS-DOS Programmer's Reference* for DOS 5.0 incorrectly documents this interrupt as superseded.

**See also:** INT 21h Function 34h (above), Network INT 2Ah Function 84h (NI), Windows INT 2Fh Function 1680h (Chapter 43)

---

### INT 29h — DOS 2+
### *FAST CONSOLE OUTPUT* — *Callout*

| Registers at call: | Return Registers: |
|---|---|
| AL = character to display | none |

**Details:** This function is automatically called when writing to a device with bit 4 of its device driver header set (see also Table 24-83).

COMMAND.COM v3.2 and v3.3 compare the INT 29h vector against the INT 20h vector and assume that ANSI.SYS is installed if the segment is larger.

The default handler under DOS 2.x and 3.x simply calls INT 10h Function 0Eh.

The default handler under DESQview 2.2 understands the <Esc>[2J screen-clearing sequence, calls INT 10h Function 0Eh for all others.

**See also:** INT 21h Function 52h (above), INT 2Fh Function 0802h (above), AVATAR.SYS INT 79h (Chapter 13)

---

### INT 2Bh to INT 2Dh — DOS 2+
### *RESERVED*

**Details:** These vectors are not used in DOS versions through 6.10, and point at IRET instructions.

---

### INT 30h — DOS 1+
### *FAR JMP INSTRUCTION FOR CP/M-STYLE CALLS (NOT A VECTOR!)*

**Purpose:** The CALL 5 entry point does a FAR jump to absolute address 000C0h (the INT 30h vector), which contains another FAR jump to the DOS kernel's CP/M-compatible entry point.

**BUG:** Under DOS 2+, the instruction at PSP:0005h points two bytes too low in memory.

**See also:** INT 21h Function 26h (above)

---

### INT 31h — *overwritten by CP/M jump instruction in INT 30h*

---

### INT 6Ch — DOS 3.2
### *Realtime Clock update*

## CHAPTER 25

# *European MS-DOS 4.00*

European MS-DOS 4.0 was written for Siemens in Germany and then used by several other European OEMs; its release falls between mainstream versions 3.2 and 3.3. Because of the existence of this version, the MS-DOS corresponding to PC-DOS versions 4.0x was labeled version 4.01.

See Function 87h for an installation check.

## Kernel

### INT 21h Function 59h, Subfunction 0001h *European MS-DOS 4.0*

### *GET HARD ERROR INFORMATiON*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 59h | ES:DI -> hard error information packet |
| BX = 0001h | (see Table 25-1) for most recent hard (critical) error |

**See also:** Function 95h, MS-DOS Function 59h (Chapter 24), MS-DOS INT 24h (Chapter 24)

**25-1 Format of hard error information packet:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | contents of AX at system entry |
| 02h | WORD | Process ID which encountered error |
| 04h | WORD | contents of AX at time of error |
| 06h | BYTE | error type<br>00h physical I/O error<br>01h disk change request<br>02h file sharing violation<br>03h FCB problem<br>04h file locking violation<br>05h bad FAT<br>06h network detected error |
| 07h | BYTE | INT 24h error code |
| 08h | WORD | extended error code (see Table 24-2) |
| 0Ah | DWORD | pointer to associated device |

### INT 21h Function 80h *European MS-DOS 4.0*

### *"AEXEC" - EXECUTE PROGRAM IN BACKGROUND*

**Purpose:** Asynchronously execute a program, creating a new process for it.

**Registers at call:**

AH = 80h

CX = mode
  0000h place child in zombie mode on exit to preserve exit code
  0001h discard child process and exit code on termination
DS:DX -> ASCIZ full program name
ES:BX -> parameter block (see Table 24-42)

**Return Registers:**
CF clear if successful
  AX = Command Subgroup ID (CSID)
CF set on error
  AX = error code (see Table 24-2)

**Details:** This function is called by the DETACH command.

There is a system-wide limit of 32 processes.

The CSID is used to identify all processes that have been spawned by a given process, whether directly or indirectly.

Programs to be run in the background must use the new executable format (see Function 4Bh in Chapter 24). The "NE" new executable format made its first appearance in European MS-DOS 4.0.

Background processes may only perform asynchronous (background) EXECs, either this function or Function 4B04h (see Chapter 24).

Background processes may execute INT 11h, INT 12h, INT 21h, INT 2Ah, and INT 2Fh at any time; they may execute INT 10h and INT 16h only while they have opened a popup screen via INT 2Fh Function 1401h (below); no other interrupts may be executed from the background.

Background processes may not use drive B: or overlay their code segments.

**Conflicts:** PCW Weather Card (Chapter 66)

**See also:** Function 87h, POPUP INT 2Fh Function 1400h (below), MS-DOS Function 4Bh (Chapter 24)

---

### INT 21h Function 81h

*European MS-DOS 4.0*

### *"FREEZE" - STOP A PROCESS*

**Purpose:** Temporarily suspend a process or a process and all of its children.

**Registers at call:**
AH = 81h
BX = flag (0000h freeze command subtree, 0001h only specified process)
CX = Process ID of head of command subtree

**Return Registers:**
CF clear if successful
CF set on error
  AX = error code (no such process)

**Details:** If BX=0001h, this call will not return until the process is actually frozen, which may not be until after it unblocks from an I/O operation.

**See also:** Functions 82h, 89h, and 8E00h; TopView INT 15h Function 101Dh (Chapter 44)

---

### INT 21h Function 82h

*European MS-DOS 4.0*

### *"RESUME" - RESTART A PROCESS*

**Purpose:** Restart a previously-suspended process or a process and all of its children.

**Registers at call:**
AH = 82h
BX = flag (0000h resume command subtree, 0001h only specified process)
CX = Process ID of head of command subtree

**Return Registers:**
CF clear if successful
CF set on error
  AX = error code (no such process)

**See also:** Function 81h, TopView INT 15h Function 101Eh (Chapter 44)

---

### INT 21h Function 83h

*European MS-DOS 4.0*

### *"PARTITION" - GET/SET FOREGROUND PARTITION SIZE*

**Purpose:** Specify or determine how much memory may be allocated by the foreground process.

**Registers at call:**
AH = 83h
AL = function
  00h get size
  01h set new size
    BX = new size in paragraphs

**Return Registers:**
CF clear if successful
  BX = current size (function 00h) or old size (function 01h)
CF set on error
  AX = error code (01h,07h,0Dh) (see Table 24-2)

**Details:** If the partition size is set to 0000h, no partition management is done and all memory allocation is compatible with DOS 3.2.

The partition size can be changed regardless of what use is being made of the changed memory; subsequent allocations will follow the partition rules (foreground processes may allocate only foreground memory; background processes allocate background memory first, then foreground memory).

**Conflicts:** Virus (Chapter 59)
**See also:** MS-DOS Functions 48h and 4Ah (Chapter 24)

---

### INT 21h Function 8400h

*European MS-DOS 4.0*

### *"CREATMEM" - CREATE A SHARED MEMORY AREA*

**Purpose:** Create an area of memory which may be accessed by multiple processes.

**Registers at call:**
AX = 8400h
BX = size in bytes (0000h = 65536)
CX = flags
  bit 6: zero-initialize segment
DS:DX -> ASCIZ name (must begin with "\SHAREMEM\")

**Return Registers:**
CF clear if successful
  AX = segment address of shared memory global object
CF set on error
  AX = error code (06h,08h) (see Table 24-2)

**Details:** Shared memory objects are created as special files (thus the restriction on the name).

On successful creation, the reference count is set to 1.

**Conflicts:** Virus (Chapter 59)
**See also:** Functions 8401h and 8402h, DESQview INT 15h Function DE19h (Chapter 44)

---

### INT 21h Function 8401h

*European MS-DOS 4.0*

### *"GETMEM" - OBTAIN ACCESS TO SHARED MEMORY AREA*

**Purpose:** Get the address of a previously-created area of memory which may be accessed by multiple processes.

**Registers at call:**
AX = 8401h
CX = flags
  bit 7: writeable segment (ignored by MS-DOS 4.0)
DS:DX -> ASCIZ name (must begin with "\SHAREMEM\")

**Return Registers:**
CF clear if successful
  AX = segment address of shared memory global object
  CX = size in bytes
CF set on error
  AX = error code (invalid name)

**Details:** This call increments the reference count for the shared memory area.
**Conflicts:** Virus (Chapter 59)
**See also:** Functions 8400h and 8402h

---

### INT 21h Function 8402h

*European MS-DOS 4.0*

### *"RELEASEMEM" - FREE SHARED MEMORY AREA*

**Purpose:** Indicate that the specified area of shared memory will no longer be used by the caller.

**Registers at call:**
AX = 8402h
BX = handle (segment address of shared memory object)

**Return Registers:**
CF clear if successful
CF set on error
  AX = error code (no such name)

**Details:** The reference count is decremented and the shared memory area is deallocated if the new reference count is zero.
**Conflicts:** Virus (Chapter 59)
**See also:** Functions 8400h and 8401h, DESQview INT 15h Function DE19h (Chapter 44)

---

### INT 21h Function 85h
### *UNDETERMINED FUNCTION*

*European MS-DOS 4.0 (undocumented)*

**Registers at call:**
AH = 85h
*others, if any, unknown*

**Return Registers:**
*unknown*

---

### INT 21h Function 86h
### *"SETFILETABLE" - INSTALL NEW FILE HANDLE TABLE*

*European MS-DOS 4.0*

**Purpose:** Adjust the size of the per-process open file table, thus raising or lowering the limit on the number of files the caller can open simultaneously.

**Registers at call:**
AH = 86h
BX = total number of file handles in new table

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (06h,08h) (see Table 24-2)

**Details:** Any currently-open files are copied to the new table.

If the table is increased beyond the default 20 handles, only the first 20 will be inherited by child processes.

Error 06h is returned if the requested number of handles exceeds system limits or would require closing currently-open files.

**See also:** MS-DOS Functions 26h and 67h (Chapter 24)

---

### INT 21h Function 87h
### *"GETPID" - GET PROCESS IDENTIFIER*

*European MS-DOS 4.0*

**Purpose:** Determine an identifier by which to access the calling process.

**Registers at call:**
AH = 87h

**Return Registers:**
AX = PID
BX = parent process's PID
CX = Command Subgroup ID (CSID)

**Details:** Called by MS C v5.1 getpid() function.

This function apparently must return AX=0001h for INT 21h Function 80h to succeed.

One possible check for European MS-DOS 4.0 is to issue this call with AL=00h and check whether AL is nonzero on return.

**See also:** Function 80h, MS-DOS Functions 30h and 62h (Chapter 24)

---

### INT 21h Function 88h
### *UNDETERMINED FUNCTION*

*European MS-DOS 4.0 (undocumented)*

**Registers at call:**
AH = 88h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 87h

---

### INT 21h Function 89h
### *SLEEP*

*European MS-DOS 4.0*

**Purpose:** Suspend the calling process for the specified duration.

**Registers at call:**
AH = 89h
CX = time in milliseconds or 0000h to give up time slice

**Return Registers:**
CF clear if successful
CX = 0000h
CF set on error

AX = error code (interrupted system call)
CX = sleep time remaining

**Details:** The sleep interval is rounded up to the next higher increment of the scheduler clock, and may be extended further if other processes are running.

This call may be interrupted by signals (see Function 8Dh).

Reportedly called by Microsoft C 4.0 startup code.

Background processes have higher priority than the foreground process, and should thus periodically yield the CPU.

**Conflicts:** Virus (Chapter 59)

**See also:** Function 81h, TopView INT 15h Function 1000h (Chapter 44), MS Windows INT 2Fh Function 1680h (Chapter 43), NetWare INT 7Ah Function 000Ah (NI)

---

### INT 21h Function 8Ah

*European MS-DOS 4.0*

### *"CWAIT" - WAIT FOR CHILD TO TERMINATE*

**Purpose:** Get the return code from an asynchronously-executed child program, optionally waiting if no return code is available yet.

**Registers at call:**
AH = 8Ah
BL = range (00h command subtree, 01h any child)
BH = suspend flag
    00h suspend if children exist but none is dead
    01h return if no dead children
CX = Process ID of head of command subtree

**Return Registers:**
CF clear if successful
    AH = termination type
        00h normal termination
        01h aborted by Control-C
        02h aborted by I/O error
        03h terminate and stay resident
        04h aborted by signal
        05h aborted by program error
    AL = return code from child or aborting signal
    BX = PID of child (0000h if no dead children)
CF set on error
    AX = error code (no child,interrupted system call)

**See also:** Functions 80h and 8Dh, MS-DOS Functions 4Bh and 4Dh (Chapter 24)

---

### INT 21h Function 8Bh

*European MS-DOS 4.0*
*(undocumented)*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AH = 8Bh
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 87h

---

### INT 21h Function 8Ch

*European MS-DOS 4.0*

### *SET SIGNAL HANDLER*

**Purpose:** Specify the routine which will be invoked on a number of exceptional conditions.

**Registers at call:**
AH = 8Ch
AL = signal number (see Table 25-2)
BL = action (see Table 25-3)
DS:DX -> signal handler (see Table 25-4)

**Return Registers:**
CF clear if successful
    AL = previous action
    ES:BX -> previous signal handler
CF set on error
    AX = error code (01h,invalid SigNumber or Action) (see Table 24-2)

**Details:** All signals will be sent to the most recently installed handler.

**See also:** Function 8Dh

**25-2 Values for signal number:**

| | | |
|---|---|---|
| 01h | SIGINTR | Control-C or user defined interrupt key |
| 08h | SIGTERM | program termination |
| 09h | SIGPIPE | broken pipe |
| 0Dh | SIGUSER1 | reserved for user definition |
| 0Eh | SIGUSER2 | reserved for user definition |

**25-3 Values for signal action:**

| | | |
|---|---|---|
| 00h | SIG_DFL | terminate process on receipt |
| 01h | SIG_IGN | ignore signal |
| 02h | SIG_GET | signal is accepted |
| 03h | SIG_ERR | sender gets error |
| 04h | SIG_ACK | acknowledge received signal and clear it, but don't change current setting |

**25-4 Signal handler calling convention:**

**Registers at call:**
AL = signal number
AH = signal argument

**Return Registers:**
RETF, CF set: terminate process
RETF, CF clear, ZF set: abort any interrupted system call with an error
RETF, CF clear, ZF clear: restart any interrupted system call
IRET: restart any interrupted system call

**Details:** The signal handler may also perform a nonlocal GOTO by resetting the stack pointer and jumping; before doing so, it should dismiss the signal by calling this function with BL=04h.

---

**INT 21h Function 8Dh** *European MS-DOS 4.0*

### *SEND SIGNAL*

**Purpose:** Invoke the exceptional-condition handler for the specified process.

**Registers at call:**
AH = 8Dh
AL = signal number (see Table 25-2)
BH = signal argument
BL = action
    00h send to entire command subtree
    01h send only to specified process
DX = Process ID

**Return Registers:**
CF clear if successful
CF set on error
    AX = error code (01h,06h) (see Table 24-2)

**Details:** Error 06h may be returned if one or more of the affected processes have an error handler for the signal.

**See also:** Function 8Ch

---

**INT 21h Function 8E00h** *European MS-DOS 4.0*

### *"SETPRI" - GET/SET PROCESS PRIORITY*

**Purpose:** Specify or determine the execution priority of the specified process or the process and all of its children.

**Registers at call:**
AX = 8E00h
BH = 00h
BL = action
    00h set priority for command subtree
    01h set priority for specified process only
CX = Process ID
DH = 00h
DL = change in priority (00h to get priority)

**Return Registers:**
CF clear if successful
    DL = process priority
    DH destroyed
CF set on error
    AX = error code (01h,no such process) (see Table 24-2)

**See also:** Function 81h

---

### INT 21h Functions 8Fh to 92h

*European MS-DOS 4.0 (undocumented)*

***UNDETERMINED FUNCTIONS***

| Registers at call: | Return Registers: |
|---|---|
| AH = 8Fh to 92h<br>*others, if any, unknown* | *unknown* |

**Conflicts:** Virus Function 90h (Chapter 59)
**See also:** Function 87h

---

### INT 21h Function 93h

*European MS-DOS 4.0*

***"PIPE" - CREATE A NEW PIPE***

**Purpose:** Create a communications channel which may be used for interprocess data and command exchanges.

| Registers at call: | Return Registers: |
|---|---|
| AH = 93h<br>CX = size in bytes | CF clear if successful<br>AX = read handle<br>BX = write handle<br>CF set on error<br>AX = error code (08h) (see Table 24-2) |

**See also:** Function 84h; MS-DOS Functions 3Ch, 3Fh, and 40h (Chapter 24)

---

### INT 21h Function 94h

*European MS-DOS 4.0 (undocumented)*

***UNDETERMINED FUNCTION***

| Registers at call: | Return Registers: |
|---|---|
| AH = 94h<br>*others, if any, unknown* | *unknown* |

**See also:** Function 87h

---

### INT 21h Function 95h

*European MS-DOS 4.0*

***HARD ERROR PROCESSING***

**Purpose:** Specify whether hard (critical) errors should automatically fail the system call or invoke an INT 24h.

| Registers at call: | Return Registers: |
|---|---|
| AH = 95h<br>AL = new state<br>00h enabled<br>01h disabled, automatically fail hard errors | AX = previous setting |

**See also:** INT 24h (Chapter 24)

---

### INT 21h Functions 96h to 98h

*European MS-DOS 4.0 (undocumented)*

***UNDETERMINED FUNCTION***

| Registers at call: | Return Registers: |
|---|---|
| AH = 96h to 98h<br>*others, if any, unknown* | *unknown* |

**Conflicts:** Virus Function 9753h (Chapter 59)

---

### INT 21h Function 99h

*European MS-DOS 4.0 (partially doc.)*

***"PBLOCK" - BLOCK A PROCESS***

**Purpose:** Suspend the calling process until another process sends a "restart" signal or a timeout occurs.

**Registers at call:**
AH = 99h
DS:BX -> memory location on which to block

CX = timeout in milliseconds
DH = nonzero if interruptible

**Return Registers:**
CF clear if awakened by event
    AX = 0000h
CF set if unusual wakeup
    ZF set if timeout, clear if interrupted by signal
    AX = nonzero

**See also:** Function 9Ah, DRIVER.SYS INT 2Fh Function 0802h (Chapter 24)

---

### INT 21h Function 9Ah
### *"PRUN" - UNBLOCK A PROCESS*

*European MS-DOS 4.0 (partially doc.)*

**Purpose:** Restart all processes waiting for the specified "restart" signal.

**Registers at call:**
AH = 9Ah
DS:BX -> memory location on which processes may have blocked

**Return Registers:**
AX = number of processes awakened
ZF set if no processes awakened

**See also:** Function 99h, INT 2Fh Function 0802h (Chapter 24)

## POPUP Utility

---

### INT 2Fh Function 1400h
### *"CheckPu" - INSTALLATION CHECK*

*European MS-DOS 4.0 POPUP*

**Registers at call:**
AX = 1400h

**Return Registers:**
AX = FFFFh if installed
    BX = maximum memory required to save screen and keyboard info
CF clear if successful
CF set on error
    AX = error code
        0002h invalid function
        0004h unknown error

**Details:** The POPUP interface is used by background programs (see INT 21h Function 80h above) to communicate with the user.

**Conflicts:** NLSFUNC (Chapter 27)

**See also:** Functions 1401h to 1403h

---

### INT 2Fh Function 1401h
### *"PostPu" - OPEN/CLOSE POPUP SCREEN*

*European MS-DOS 4.0 POPUP*

**Registers at call:**
AX = 1401h
DL = function (00h open, 01h close)
DH = wait flag
    00h block until screen opens
    01h return error if screen is not available
    02h urgent--always open screen immediately

**Return Registers:**
CF clear if successful
    BX = amount of memory needed to save screen and keyboard info, 0000h if default save location can be used (only if DH was 02h)
CF set on error

**Details:** The application using the screen is frozen until the popup screen is closed.

**Conflicts:** NLSFUNC (Chapter 27)

**See also:** Functions 1400h, 1402h, and 1403h

---

**INT 2Fh Function 1402h** *European MS-DOS 4.0 POPUP*

***"SavePu" - SAVE POPUP SCREEN***

**Registers at call:**
AX = 1402h
ES:DI -> save buffer (0000h:0000h for default buffer in POPUP)

**Return Registers:**
CF clear if successful
CF set on error
  AX = error code
    0001h process does not own screen
    0004h unknown error
    0005h invalid pointer

**Conflicts:** NLSFUNC (Chapter 27)
**See also:** Functions 1400h, 1401h, and 1403h

---

**INT 2Fh Function 1403h** *European MS-DOS 4.0 POPUP*

***"RestorePu" - RESTORE SCREEN***

**Registers at call:**
AX = 1403h
ES:DI -> buffer containing saved screen (0000h:0000h for default buffer in POPUP)

**Return Registers:**
CF clear if successful
CF set on error
  AX = error code (see Function 1402h)

**Conflicts:** NLSFUNC (Chapter 27)
**See also:** Functions 1400h, 1401h, and 1402h

## CHAPTER 26

# *MS-DOS Compatibles*

While MS-DOS itself remains the *de facto* standard operating system for the Intel 80x86 line of CPU chips, a number of compatible and semi-compatible competitors have become significant. Among these are DR-DOS, created by Digital Research but now owned by Novell, and IBM's OS/2. This chapter is devoted to such compatible systems. Additional information appears in Chapters 47 and 48.

## Digital Research / Novell DOSes

Before MS-DOS came into existence, the most widely used operating system for microcomputers was almost certainly CP/M, from Digital Research. The 16-bit version of CP/M failed to find favor, though, and DR almost vanished from the operating/system scene so far as a casual observer could determine. The company, however, never did abandon the field. In the wake of the MS-DOS/PC-DOS Version 4.0 fiasco, they brought out a compatible system that did most of the things users had expected, as DR-DOS 5.0.

When MS-DOS Version 5.0 appeared, correcting virtually all of the problems users had experienced with version 4.0, DR countered with DR-DOS 6.0 and later 6.1. Novell's purchase of DR in 1992 added strength to the offerings, and as this is written Novell DOS 7 is promised for delivery in "summer 1993" but has not yet arrived in public view. This section discusses the API features of all known versions of DR-DOS that differ from those in MS-DOS.

In addition to the calls covered here, DR-DOS's TaskMAX task switcher API is presented in Chapter 47. The CP/M-86 compatible interface provided by Digital Research's multitasking operating systems (Concurrent DOS and DR MultiUser DOS) is detailed in Chapter 48.

---

### INT 21h Function 4302h

*DR-DOS 3.41+ internal*

***GET ACCESS RIGHTS***

**Purpose:** Determine which operations the calling program may perform on a specified file without being required to provide a password.

**Registers at call:**
AX = 4302h
DS:DX -> ASCIZ pathname

**Return Registers:**
CF clear if successful
    CX = access rights (see Table 26-1)
    AX = CX (DR-DOS 5.0)
CF set on error
    AX = error code

**Details:** This protection scheme has been coordinated on all current Digital Research/Novell operating systems (DR-DOS 3.41+, DRMDOS 5.x, and FlexOS 2+).

This function is documented in DR-DOS 6.0 and corresponds to the "Get/Set File Attributes" function, subfunction 2, documented in Concurrent DOS.

Only FlexOS actually uses the "execution" bits; DR-DOS 3.41+ treats them as "read" bits.

DR-DOS 3.41-5.x only use bits 0-3. Only DR-DOS 6.0 using a DRMDOS 5.x security system allowing for users and groups uses bits 4-11.
**See also:** Function 4303h

### 26-1 Bitfields for access rights:

| | |
|---|---|
| bit 0 | owner delete requires password |
| bit 1 | owner execution requires password (FlexOS) |
| bit 2 | owner write requires password |
| bit 3 | owner read requires password |
| bit 4 | group delete requires password |
| bit 5 | group execution requires password (FlexOS) |
| bit 6 | group write requires password |
| bit 7 | group read requires password |
| bit 8 | world delete requires password |
| bit 9 | world execution requires password (FlexOS) |
| bit 10 | world write requires password |
| bit 11 | world read requires password |

---

## INT 21h Function 4303h

*DR-DOS 3.41+ internal*

### *SET ACCESS RIGHTS AND PASSWORD*

**Registers at call:**
AX = 4303h
CX = access rights
  bits 11-0: access rights (see Table 26-1)
  bit 15: new password is to be set
DS:DX -> ASCIZ pathname
[DTA] = new password if CX bit 15 is set (blank-padded to 8 characters)

**Return Registers:**
CF clear if successful
CF set on error
  AX = error code

**Details:** If the file is already protected, the old password must be added after the pathname, separated by a ";".

This function is documented in DR-DOS 6.0 and corresponds to the "Get/ Set File Attributes" function, subfunction 3, documented in Concurrent DOS.
**See also:** Functions 4305h and 4454h, MS-DOS Functions 0Fh, 17h, and 4302h (Chapter 24)

---

## INT 21h Function 4304h

*DR-DOS 5.0-6.0 internal (undocumented)*

### *GET ENCRYPTED PASSWORD*

**Registers at call:**
AX = 4304h
DS:DX -> ASCIZ filename
*others, if any, unknown*

**Return Registers:**
CF clear if successful
  CX = AX = *unknown*
CF set on error
  AX = error code (see Table 24-2)

**Details:** This function is only supported by DR-DOS 5.0 and 6.0 and DRMDOS 5.1.
**See also:** Functions 4303h and 4305h

---

## INT 21h Function 4305h

*DR-DOS 5.0-6.0 internal (undocumented)*

### *SET EXTENDED FILE ATTRIBUTES*

**Purpose:** This function allows the extended attributes, and optionally the encrypted password, of a file to be set.

**Registers at call:**
AX = 4305h
DS:DX -> ASCIZ filename
*others, if any, unknown*

**Return Registers:**
CF clear if successful
CF set on error
  AX = error code (see Table 24-2)

**Details:** This function is only supported by DR-DOS 5.0 and 6.0 and DRMDOS 5.1.
**See also:** Function 4304h, Banyan VINES Function 4311h (NI)

### INT 21h Function 4306h

*DR-DOS 6.0*

*GET FILE OWNER*

**Registers at call:**
AX = 4306h
DS:DX -> ASCIZ filename

**Return Registers:**
CF clear if successful
    AX = CX = value set with Function 4307h
CF set on error
    AX = error code (see Table 24-2)

**See also:** Function 4307h

### INT 21h Function 4307h

*DR-DOS 6.0*

*SET FILE OWNER*

**Registers at call:**
AX = 4307h
CX = *owner identification number?*
DS:DX -> ASCIZ filename

**Return Registers:**
CF clear if successful
CF set on error
    AX = error code (see Table 24-2)

**See also:** Function 4306h

### INT 21h Function 4380h

*Novell DOS 7*

*UNDELETE PENDING DELETE FILE*

**Registers at call:**
AX = 4380h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 4381h, MS-DOS Function 41h (Chapter 24)

### INT 21h Function 4381h

*Novell DOS 7*

*PURGE PENDING DELETE FILE*

**Registers at call:**
AX = 4381h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 4380h, MS-DOS Function 41h (Chapter 24)

### INT 21h Functions 4412h to 4418h

*DR-DOS 5.0*

*OBSOLETE FUNCTIONS*

**Registers at call:**
AH = 4412h, 4414h, 4416h, 4417h, 4418h
others vary by function

**Return Registers:**
vary by function

**Details:** These obsolete subfunctions (which will not be supported in future versions of DR-DOS) are identical to Functions 4452h through 4458h.

**See also:** Functions 4452h to 4458h

### INT 21h Function 4451h

*Concurrent DOS v3.2+*

*INSTALLATION CHECK*

**Registers at call:**
AX = 4451h

**Return Registers:**
CF set if not Concurrent DOS
    AX = error code (see Table 24-2)
CF clear if successful
    AH = single-user/ multiuser nature
        10h single-user
            AL = operating system version ID (see Table 26-3)
        14h multiuser
            AL = operating system version ID (see Table 26-2)

**Details:** As of Concurrent DOS/ XM 5.0 (possibly earlier), the version is stored in the environment variable VER.

Use this function if you are looking for multiuser capabilities, Function 4452h for single-user. This function should never return the single-user values.

**See also:** Functions 4452h and 4459h

**26-2 Multiuser operating system version IDs:**

| | |
|---|---|
| 32h | Concurrent PC DOS 3.2 |
| 41h | Concurrent DOS 4.1 |
| 50h | Concurrent DOS/ XM 5.0 or Concurrent DOS/ 386 1.1 |
| 60h | Concurrent DOS/ XM 6.0 or Concurrent DOS/ 386 2.0 |
| 62h | Concurrent DOS/ XM 6.2 or Concurrent DOS/ 386 3.0 |
| 66h | DR Multiuser DOS 5.1 |
| 67h | Concurrent DOS 5.1 |

## INT 21h Function 4452h

*DR-DOS 3.41+*

### *DETERMINE DOS TYPE/GET DR-DOS VERSION*

**Registers at call:**
AX = 4452h ("DR")
CF set

**Return Registers:**
CF set if not DR-DOS
 AX = error code (see Table 24-2)
CF clear if DR-DOS
 DX = AX = version code
 AH = single-user/ multiuser nature
  10h single-user
   AL = operating system version ID (see Table 26-3)
  14h multiuser
   AL = operating system version ID (see Table 26-2)

**Details:** The DR-DOS version is stored in the environment variable VER.

Use this function if looking for single-user capabilities, Function 4451h if looking for multiuser; this call should never return multiuser values.

**See also:** Functions 4412h, 4451h, and 4459h

**26-3 Single-tasking operating system version IDs:**

| | |
|---|---|
| 60h | DOS Plus |
| 63h | DR-DOS 3.41 |
| 64h | DR-DOS 3.42 |
| 65h | DR-DOS 5.00 |
| 67h | DR-DOS 6.00 |
| 70h | PalmDOS |
| 71h | DR-DOS 6.0 March 1993 "business update" |
| 72h | Novell DOS 7.0 |

## INT 21h Function 4454h

*DR-DOS 3.41+*
*(undocumented)*

### *SET GLOBAL PASSWORD*

**Purpose:** Specify the master password for accessing files.

**Registers at call:**
AX = 4454h
DS:DX -> password string (blank-padded to 8 characters)

**Return Registers:**
none

**See also:** Functions 4303h and 4414h

### INT 21h Function 4456h
### *HISTORY BUFFER CONTROL*

*DR-DOS 5.0+*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4456h | AL = *unknown*--20h if DL bit 0 set, A0h if clear (DR-DOS 6.0) |
| DL = flag | |
| bit 0: buffer type | |
| set = COMMAND.COM history buffers | |
| clr = application | |

**Details:** The DR-DOS 6.0 COMMAND.COM calls this function.

### INT 21h Function 4457h
### *SHARE/HILOAD CONTROL*

*DR-DOS 5.0+*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4457h | |
| DH = subfunction | |
| 00h enable/ disable SHARE | |
| DL = 00h disable, 01h enable | |
| DL = other | |
| Return: AX = *unknown* | |
| 01h get HILOAD status | AX = status (0000h off, 0001h on) |
| 02h set HILOAD status | AX = *unknown* |
| DL = new state (00h off, 01h on) | |
| other | AX = *unknown* |

**Details:** The DR-DOS 6.0 COMMAND.COM calls this function.
**See also:** Function 4457h Subfunction FFFFh

### INT 21h Function 4457h, Subfunction FFFFh
### *GET SHARE STATUS*

*DR-DOS 6.0*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4457h | AX = SHARE status |
| DX = FFFFh | |

**See also:** SHARE INT 2Fh Function 1000h (Chapter 27)

### INT 21h Function 4458h
### *GET POINTER TO INTERNAL VARIABLE TABLE*

*DR-DOS 5.0+ internal*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4458h | ES:BX -> internal variable table (see Table 26-4) |
| | AX = *unknown* (0B50h for DR-DOS 5.0, 0A56h for DR-DOS 6.0) |

**See also:** Function 4452h

**26-4 Format of internal variable table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to *unknown item* |
| 04h | 7 BYTEs | *unknown* |
| 0Bh | WORD | KB of extended memory at startup |
| 0Dh | BYTE | number of far jump entry points |
| 0Eh | WORD | segment containing far jumps to DR-DOS entry points (see Table 26-5) |

**---DR-DOS 6.0---**

| | | |
|---|---|---|
| 10h | WORD | (only if kernel loaded in HMA) offset in HMA of first free HMA memory block (see Table 26-6) or 0000h if none; segment is FFFFh |
| 12h | WORD | pointer to segment of environment variables set in CONFIG, or 0000h if already used |
| 14h | WORD | (only if kernel loaded in HMA) offset in HMA of first used HMA memory block (see Table 26-6) or 0000h if none; segment is FFFFh |

**Note:** the segment used for the DR-DOS 6.0 CONFIG environment variables (excluding COMSPEC, VER and OS) is only useful for programs/ drivers called from CONFIG.SYS. The word is set to zero later and the area lost.

### 26-5 Format of kernel entry jump table for DR-DOS 5.0-6.0:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 5 BYTEs | far jump to kernel entry point for CP/ M CALL 5 |
| 05h | 5 BYTEs | far jump to kernel entry point for INT 20h |
| 0Ah | 5 BYTEs | far jump to kernel entry point for INT 21h |
| 0Fh | 5 BYTEs | far jump to kernel entry point for *INT 22h* (RETF) |
| 14h | 5 BYTEs | far jump to kernel entry point for *INT 23h* (RETF) |
| 19h | 5 BYTEs | far jump to kernel entry point for INT 24h |
| 1Eh | 5 BYTEs | far jump to kernel entry point for INT 25h |
| 23h | 5 BYTEs | far jump to kernel entry point for INT 26h |
| 28h | 5 BYTEs | far jump to kernel entry point for INT 27h |
| 2Dh | 5 BYTEs | far jump to kernel entry point for INT 28h |
| 32h | 5 BYTEs | far jump to kernel entry point for INT 2Ah (IRET) |
| 37h | 5 BYTEs | far jump to kernel entry point for INT 2Bh (IRET) |
| 3Ch | 5 BYTEs | far jump to kernel entry point for INT 2Ch (IRET) |
| 41h | 5 BYTEs | far jump to kernel entry point for INT 2Dh (IRET) |
| 46h | 5 BYTEs | far jump to kernel entry point for INT 2Eh (IRET) |
| 4Bh | 5 BYTEs | far jump to kernel entry point for INT 2Fh |

**Note:** All of these entry points are indirected through this jump table to allow the kernel to be relocated into high memory while leaving the actual entry addresses in low memory for maximum compatibility.

### 26-6 Format of HMA Memory Block (DR-DOS 6.0 kernel loaded in HMA):

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | offset of next HMA Memory Block (0000h if last block) |
| 02h | WORD | size of this block in bytes (at least 10h) |
| 04h | BYTE | type of HMA Memory Block (interpreted by MEM)<br>00h system<br>01h KEYB<br>02h NLSFUNC<br>03h SHARE<br>04h TaskMAX<br>05h COMMAND |
| 05h | var | TSR (or system) code and data. DR-DOS TSR's, such as KEYB, hooks interrupts using segment FFFEh instead FFFFh. |

---

### INT 21h Function 4459h
*DR MultiUser DOS 5.0*

### *API*

**Registers at call:**
AX = 4459h
CL = function (see INT E0h in Chapter 48)
DS,DX = parameters

**Return Registers:**
varies by function

**Details:** DR-DOS 5.0 returns CF set and AX=0001h.
This API is also available on INT E0h.

**See also:** Functions 4452h and E0h, CP/ M-86 INT E0h (Chapter 48)

---

### INT 21h Function 69h
*DR-DOS 5.0*

### *NULL FUNCTION*

**Purpose:** This function does nothing and returns immediately.

**Registers at call:**
AH = 69h

**Return Registers:**
AL = 00h

**Conflicts:** MS-DOS (Chapter 24)

---

### INT 21h Function E0h
*Digital Research DOS Plus*

### *CALL BDOS*

**Registers at call:**
AH = E0h
CL = BDOS function number (see INT E0h in Chapter 48) other registers as appropriate for function

**Return Registers:**
as appropriate for function

**Conflicts:** OS/ 286 (Chapter 34), DoubleDOS (Chapter 49), Viruses (Chapter 59), Novell NetWare (NI)

**See also:** Function 4459h, CP/ M-86 INT E0h (Chapter 48)

---

### INT 2Fh Function 10FFh
*DR-DOS 5.0*
*(undocumented)*

### *UNDETERMINED FUNCTION*

**Purpose:** This function sets a pointer in the kernel.

**Registers at call:**
AX = 10FFh
ES:BX -> *unknown*

**Return Registers:**
none

---

### INT 2Fh Function 12FFh, Subfunction 0006h
*DR-DOS 6*
*(undocumented)*

### *VIDEO MEMORY SPACE CONTROL*

**Registers at call:**
AX = 12FFh
BX = 0006h
DX = 0000h
CX = function
  0000h get status of video memory space (MEMMAX / V)
  0001h map memory into video memory space (MEMMAX +V)
  0002h unmap memory from video memory space (MEMMAX -V)

**Return Registers:**
CF clear if successful
  AX = 0000h (successful)
  BX = segment of reserved video RAM
  CX = segment of used video RAM
  DX = segment of first upper MCB

**Details:** This functionality is provided by EMM386, and partially supported by HIDOS.SYS.

**See also:** Quarterdeck Function D201h Subfunction 4849h (Chapter 63)

---

### INT 2Fh Function 14FEh
*DR-DOS 5.0 NLSFUNC*
*(undocumented)*

### *GET EXTENDED COUNTRY INFORMATION*

**Registers at call:**
AX = 14FEh
BX = code page (FFFFh=global code

page) (see Table 24-119)
DX = country ID (FFFFh=current country) (see Table 24-!C)
ES:DI -> country information buffer
CL = info ID
  01h general internationalization info
  02h pointer to uppercase table
  04h pointer to filename uppercase table
  05h pointer to filename terminator table
  06h pointer to collating sequence table
  07h pointer to Double-Byte Character Set table
CF set (used to return error if not installed)

**Return Registers:**
CF clear if successful
  DS:SI -> requested information
CF set on error

**Details:** DR-DOS 5.0 NLSFUNC returns CF set and AX=0001h if AL was not 00h, FEh, or FFh on entry.

The DR-DOS kernel calls this function on INT 21h Function 6501h.

The value in CL is not range-checked by the DR-DOS 5.0 NLSFUNC.

**See also:** Function 14FFh, MS-DOS INT 21h Function 65h (Chapter 24)

**26-7 Format of DR-DOS COUNTRY.SYS file:**

<table>
<tr><th><i>Offset</i></th><th><i>Size</i></th><th><i>Description</i></th></tr>
<tr><td>00h</td><td>126 BYTEs</td><td>copyright notice (terminated with Ctrl-Z, padded with NULs)</td></tr>
<tr><td>7Eh</td><td>WORD</td><td>signature EDC1h</td></tr>
<tr><td>80h</td><td>var</td><td>country pointer records</td></tr>
<tr><td></td><td colspan="2"><table>
<tr><th><i>Offset</i></th><th><i>Size</i></th><th><i>Description</i></th></tr>
<tr><td>00h</td><td>WORD</td><td>country code (0000h if end of array)</td></tr>
<tr><td>02h</td><td>WORD</td><td>code page</td></tr>
<tr><td>04h</td><td>WORD</td><td><i>unknown</i> (0000h)</td></tr>
<tr><td>06h</td><td>7 WORDs</td><td>offsets in file for data tables for subfunctions 01h-07h</td></tr>
</table></td></tr>
<tr><td>var</td><td>var</td><td>country information</td></tr>
</table>

### INT 2Fh Function 14FFh
### *PREPARE CODE PAGE*

*DR-DOS 5.0 NLSFUNC*
*(undocumented)*

**Registers at call:**
AX = 14FFh
BX = code page

**Return Registers:**
AX = *unknown*
ZF set if AX=0000h

**Details:** DR-DOS 5.0 NLSFUNC returns CF set and AX=0001h if AL was not 00h, FEh, or FFh on entry.

NLSFUNC passes the codepage preparation request to each character device supporting the generic IOCTL call.

**See also:** Function 14FEh, MS-DOS INT 21h Functions 440Ch and 6602h (Chapter 24)

### INT 2Fh Function 2300h
### *INSTALLATION CHECK*

*DR-DOS 5.0 GRAFTABL*

**Registers at call:**
AX = 2300h

**Return Registers:**
AH = FFh

**Details:** This installation check does not follow the usual format of returning AL=FFh.

**See also:** Function 23h below

---

### INT 2Fh Function 23h
*DR-DOS 5.0 GRAFTABL*

### *GET GRAPHICS DATA*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 23h | AH = FFh |
| AL nonzero | ES:BX -> graphics data |

**See also:** Function 2300h

---

### INT 2Fh Function AD00h
*DR-DOS 3.41,5.0 KEYB*
*(undocumented)*

### *INSTALLATION CHECK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = AD00h | AX = FFFFh if installed |

**Conflicts:** MS-DOS DISPLAY.SYS (Chapter 27)
**See also:** MS-DOS Function AD80h (Chapter 27)

---

### INT 2Fh Function AD01h
*DR-DOS 3.41,5.0 KEYB*
*(undocumented)*

### *GET CONFIGURATION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = AD01h | BX = current code page<br>CX = current keyboard layout (0100h = US, 0102h = foreign)<br>ES = resident code segment |

**Conflicts:** MS-DOS DISPLAY.SYS (Chapter 27)
**See also:** Function AD83h

## OS/2

OS/2 (which, like PL/1, is an acronym-looking label that has no official translation) is the IBM response to Microsoft's vision of the Windows platform to provide "information at your fingertips." Originally promoted as the logical evolution of Windows, it became an independent operating system when the close working alliance between IBM and Microsoft was (at least momentarily) shattered by conflict. This section describes the OS/2 API as it affects compatibility with MS-DOS.

---

### INT 21h Function 5702h
*OS/2 v1.1+ Family API*

### *DosQFileInfo*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5702h | CF clear if successful |
| BX = file handle | CF set on error |
| CX = size of buffer for information | AX = error code |
| DX = level of information | |
| ES:DI -> buffer for information | |

**Conflicts:** MS-DOS 4.x (Chapter 24)
**See also:** Function 5702h Subfunction FFFFh, Function 5703h

---

### INT 21h Function 5702h, Subfunction FFFFh
*OS/2 v1.1+ Family API*

### *DosQPathInfo*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5702h | CF clear if successful |
| BX = FFFFh | CF set on error |
| CX = size of buffer for information | AX = error code |
| DX = level of information | |
| DS:SI -> filename | |
| ES:DI -> buffer for information | |

**Conflicts:** MS-DOS 4.x (Chapter 24)
**See also:** Function 5702h, Function 5703h Subfunction FFFFh

---

### INT 21h Function 5703h

*OS/2 v1.1+ Family API*

*DosSetFileInfo*

**Registers at call:**
AX = 5703h
BX = file handle
CX = size of information buffer
DX = level of information
ES:DI -> information buffer

**Return Registers:**
CF clear if successful
CF set on error
    AX = error code

**Conflicts:** MS-DOS 4.x (Chapter 24)
**See also:** Function 5702h, Function 5703h Subfunction FFFFh

---

### INT 21h Function 5703h, Subfunction FFFFh

*OS/2 v1.1+ Family API*

*DosSetPathInfo*

**Registers at call:**
AX = 5703h
BX = FFFFh
CX = size of information buffer
DX = level of information
DS:SI -> filename
ES:DI -> information buffer

**Return Registers:**
CF clear if successful
CF set on error
    AX = error code

**Conflicts:** MS-DOS 4.x (Chapter 24)
**See also:** Function 5702h Subfunction FFFFh, Function 5703h

---

### INT 21h Function 61h

*OS/2 v1.x FAPI*
*(undocumented)*

*OS/2 FILE SYSTEM JOIN/SUBST*

**Registers at call:**
AH = 61h
BP = 6467h ("dg")
AL = function
    00h list
    01h add
    02h delete
BX = drive number
CX = size of buffer
SI = type (0002h JOIN, 0003h SUBST)
ES:DI -> buffer

**Return Registers:**
*unknown*

**Details:** This function is used by JOIN and SUBST to communicate with the OS/2 file system. It is also supported by OS/2 v2.0+ Virtual DOS Machines.

---

### INT 21h Function 64h, Subfunction 0000h

*OS/2 v2.0+ VDM*
*(undocumented)*

*ENABLE AUTOMATIC TITLE SWITCH*

**Purpose:** Specify that OS/2 should automatically update the title of the caller's Virtual DOS Machine whenever a program is executed.

**Registers at call:**
AH = 64h
DX = 0000h (function number)
CX = 636Ch (magic value, "cl")
BX = 0000h (indicates special request)

**Return Registers:**
none

**Details:** If CX is not 636Ch on entry, INT 21h Function 6Ch is invoked, because a bug in OS/2 1.x FAPI erroneously called Function 64h instead of Function 6Ch.

**Conflicts:** MS-DOS Function 64h (Chapter 24)
**See also:** Function 64h Subfunction 0001h, MS-DOS INT 21h Function 4Bh (Chapter 24)

---

### INT 21h Function 64h, Subfunction 0001h

*OS/2 v2.0+ VDM (undocumented)*

### *SET SESSION TITLE*

**Registers at call:**
AH = 64h
DX = 0001h (function number)
CX = 636Ch (magic value, "cl")
BX = 0000h (indicates special request)
ES:DI -> new ASCIZ title (max 12 char) or "" to restore default title

**Return Registers:**
none

**Details:** If CX is not 636Ch on entry, INT 21h Function 6Ch is invoked, because a bug in OS/ 2 1.x FAPI erroneously called Function 64h instead of Function 6Ch.
**Conflicts:** MS-DOS Function 64h (Chapter 24)
**See also:** Function 64h Subfunctions 0000h and 0002h, TopView INT 15h Function 12h Subfunction 05h (Chapter 44)

---

### INT 21h Function 64h, Subfunction 0002h

*OS/2 v2.0+ VDM (undocumented)*

### *GET SESSION TITLE*

**Registers at call:**
AH = 64h
DX = 0002h (function number)
CX = 636Ch (magic value, "cl")
BX = 0000h (indicates special request)
ES:DI -> 13-byte buffer for current title

**Return Registers:**
buffer filled (single 00h if title never changed)

**Details:** If CX is not 636Ch on entry, INT 21h Function 6Ch is invoked, because a bug in OS/ 2 1.x FAPI erroneously called Function 64h instead of Function 6Ch.
**Conflicts:** MS-DOS Function 64h (Chapter 24)
**See also:** Function 64h Subfunctions 0000h and 0001h, TopView INT 15h Function 12h Subfunction 05h (Chapter 44)

---

### INT 21h Function 64h, Subfunction 0003h

*OS/2 v2.1+ VDM (undocumented)*

### *GET LASTDRIVE*

**Registers at call:**
AH = 64h
DX = 0003h (function number)
CX = 636Ch (magic value, "cl")
BX = 0000h (indicates special request)

**Return Registers:**
AL = highest drive supported

**Details:** If CX is not 636Ch on entry, INT 21h Function 6Ch is invoked, because a bug in OS/ 2 1.x FAPI erroneously called Function 64h instead of Function 6Ch.

This function is used by WinOS2.

**Conflicts:** MS-DOS Function 64h (Chapter 24)
**See also:** MS-DOS Function 52h (Chapter 24)

---

### INT 21h Function 64h, Subfunction 0004h

*OS/2 v2.1+ VDM (undocumented)*

### *GET SIZE OF PTDA JFT*

**Registers at call:**
AH = 64h
DX = 0004h (function number)
CX = 636Ch (magic value, "cl")
BX = 0000h (indicates special request)

**Return Registers:**
AX = number of entries in OS/ 2 JFT for VDM

**Details:** If CX is not 636Ch on entry, INT 21h Function 6Ch is invoked, because a bug in OS/ 2 1.x FAPI erroneously called Function 64h instead of Function 6Ch.

In an OS/2 VDM, the DOS Job File Table in the PSP contains an index into the OS/2 JFT in the Per-Task Data Area rather than an SFT index because the OS/2 SFT can contain more than 255 entries.
**Conflicts:** MS-DOS Function 64h (Chapter 24)
**See also:** MS-DOS Function 67h (Chapter 24)

---

### INT 21h Function 64h, Subfunction 0005h
### *GET SECOND SFT FLAGS WORD*

*OS/2 v2.1+ VDM*
*(undocumented)*

**Registers at call:**
AH = 64h
DX = 0005h (function number)
CX = 636Ch (magic value, "cl")
BX = 0000h (indicates special request)
DI = DOS file handle

**Return Registers:**
AX = value of second flags word from OS/2 SFT entry for file

**Details:** If CX is not 636Ch on entry, INT 21h Function 6Ch is invoked, because a bug in OS/2 1.x FAPI erroneously called Function 64h instead of Function 6Ch.

The OS/2 SFT has two flags words rather than DOS's one word, and this function provides access to the word which is not present in DOS.
**Conflicts:** MS-DOS Function 64h (Chapter 24)

---

### INT 21h Function 64h, Subfunction 0006h
### *UNLOAD DOSKRNL SYMBOLS & LOAD PROGRAM*

*OS/2 v2.1+ VDM*
*(undocumented)*

**Registers at call:**
AH = 64h
DX = 0006h (function number)
CX = 636Ch (magic value, "cl")
BX = 0000h (indicates special request)
ES:DI -> ASCIZ filespec
DS = base address for loading

**Return Registers:**
none

**Details:** If CX is not 636Ch on entry, INT 21h Function 6Ch is invoked, because a bug in OS/2 1.x FAPI erroneously called Function 64h instead of Function 6Ch.

This function is only supported by the kernel debugging version of OS2KRNL.
**Conflicts:** MS-DOS Function 64h (Chapter 24)

---

### INT 21h Function 64h, Subfunction 0007h
### *GET WinOS2 CALL GATE ADDRESS*

*OS/2 v2.1+ VDM*
*(undocumented)*

**Registers at call:**
AH = 64h
DX = 0007h (function number)
CX = 636Ch (magic value, "cl")
BX = 0000h (indicates special request)

**Return Registers:**
AX = call gate address

**Details:** If CX is not 636Ch on entry, INT 21h Function 6Ch is invoked, because a bug in OS/2 1.x FAPI erroneously called Function 64h instead of Function 6Ch.

This function is used by WinOS2 to make direct calls to OS2KRNL, bypassing the overhead of DOSKRNL.
**Conflicts:** MS-DOS Function 64h (Chapter 24)

---

### INT 21h Function 64h, Subfunction 0008h
### *GET LOADING MESSAGE*

*OS/2 v2.1+ VDM*
*(undocumented)*

**Registers at call:**
AH = 64h
DX = 0008h (function number)

CX = 636Ch (magic value, "cl")
BX = 0000h (indicates special request)

**Return Registers:**
DS:DX -> '$'-terminated message "Loading. Please wait."

**Details:** If CX is not 636Ch on entry, INT 21h Function 6Ch is invoked, because a bug in OS/2 1.x FAPI erroneously called Function 64h instead of Function 6Ch.

This function permits National Language Support for the initial message displayed while WinOS2 starts a full-screen session.

**Conflicts:** MS-DOS Function 64h (Chapter 24)

---

## INT 21h Function 64h, Subfunction 636Ch

*OS/2 v2.1+ VDM*

### *OS/2 API SUPPORT*

*(undocumented)*

**Registers at call:**
AH = 64h
CX = 636Ch ("cl")
BX = API ordinal (see Table 26-8)
other registers as appropriate for API call
if ordinal 0025h, DS:SI -> STARTDATA structure (see Table 26-8)

**Return Registers:**
as appropriate for API call
if ordinal 0025h, AX = return code

**Conflicts:** MS-DOS Function 64h (Chapter 24)

**26-8 API ordinals:**

| | |
|---|---|
| 0025h | DOS32StartSession |
| 0082h | DosGetCP |
| 00A7h | DosQFSAttach |
| 00BFh | DosEditName |
| 00CBh | DosForceDelete |
| 0144h | Dos32CreateEventSem |
| 0145h | Dos32OpenEvenSem |
| 0146h | Dos32CloseEventSem |
| 0147h | Dos32ResetEventSem |
| 0148h | Dos32PostEventSem |
| 0149h | Dos32WaitEventSem |
| 014Ah | Dos32QueryEventSem |
| 014Bh | Dos32CreateMutexSem |
| 014Ch | Dos32OpenMutexSem |
| 014Dh | Dos32CloseMutexSem |
| 014Eh | Dos32RequestMutexSem |
| 014Fh | Dos32ReleaseMutexSem |
| 0150h | Dos32QueryMutexSem |
| 0151h | Dos32CreateMuxWaitSem |
| 0152h | Dos32OpenMuxWaitSem |
| 0153h | Dos32CloseMuxWaitSem |
| 0154h | Dos32WaitMuxWaitSem |
| 0155h | Dos32AddMuxWaitSem |
| 0156h | Dos32DeleteMuxWaitSem |
| 0157h | Dos32QueryMuxWaitSem |

### 26-9 Format of STARTDATA structure:

| Offset | Size | Description |
|---|---|---|
| 00h | WORD | length of structure (must be 0018h,001Eh,0020h,0032h,or 003Ch) |
| 02h | WORD | relation of new process to caller (00h independent, 01h child) |
| 04h | WORD | fore/background (00h foreground, 01h background) |
| 06h | WORD | trace options (00h-02h, 00h = no trace) |
| 08h | DWORD | pointer to ASCIZ program title (max 62 chars) or 0000h:0000h |
| 0Ch | DWORD | pointer to ASCIZ program name (max 128 chars) or 0000h:0000h |
| 10h | DWORD | pointer to ASCIZ program args (max 144 chars) or 0000h:0000h |
| 14h | DWORD | "TermQ" (currently reserved, must be 00000000h) |
| 18h | DWORD | pointer to environment (max 486 bytes) or 0000h:0000h |
| 1Ch | WORD | inheritance (00h or 01h) |
| 1Eh | WORD | session type<br>00h OS/2 session manager determines type (default)<br>01h OS/2 full-screen<br>02h OS/2 window<br>03h PM<br>04h VDM full-screen<br>07h VDM window |
| 20h | DWORD | pointer to ASCIZ icon filename (max 128 chars) or 0000h:0000h |
| 24h | DWORD | "PgmHandle" (currently reserved, must be 00000000h) |
| 28h | WORD | "PgmControl" |
| 2Ah | WORD | initial column |
| 2Ch | WORD | initial row |
| 2Eh | WORD | initial width |
| 30h | WORD | initial height |
| 32h | WORD | reserved (0) |
| 34h | DWORD | "ObjectBuffer" (currently reserved, must be 00000000h) |
| 38h | DWORD | "ObjectBufferLen" (currently reserved, must be 00000000h) |

---

## INT 21h Function 6C01h

*OS/2 v2.0*
*(undocumented)*

### *"DosOpen2"*

**Registers at call:**
AX = 6C01h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** This function is similar to Function 6C00h, but supports OS/2's extended attributes.

**See also:** Functions 5704h and 6Fh, MS-DOS Function 6C00h (Chapter 24)

---

## INT 21h Function 6Dh

*OS/2 v1.x FAPI*
*(undocumented)*

### *"DosMkDir2"*

**Registers at call:**
AH = 6Dh
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** This function is also supported by OS/2 v2.0+ Virtual DOS Machines.

**Conflicts:** DOS 5+ ROM version (Chapter 24)

**See also:** MS-DOS Function 39h (Chapter 24)

---

**INT 21h Function 6Eh** *OS/2 v1.x FAPI (undocumented)*

***"DosENumAttrib"***

**Registers at call:**
AH = 6Eh
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** This function is also supported by OS/ 2 v2.0+ Virtual DOS Machines.
**Conflicts:** DOS 5+ ROM version (Chapter 24)
**See also:** Functions 5703h and 6Fh, Network Redirector INT 2Fh Function 112Dh (NI)

---

**INT 21h Function 6Fh** *OS/2 v1.x FAPI (undocumented)*

***"DosQMaxEASize" - GET MAXIMUM SIZE OF EXTENDED ATTR***

**Registers at call:**
AH = 6Fh
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** This function is also supported by OS/ 2 v2.0+ Virtual DOS Machines.
**Conflicts:** DOS 5+ ROM version (Chapter 24)
**See also:** Functions 5703h, 6C01h, and 6Eh

---

**INT 2Fh Function 4001h** *OS/2 compatibility box Callout*

***SWITCHING DOS TO BACKGROUND***

**Registers at call:**
AX = 4001h

**Return Registers:**
none

**Details:** This function is called by OS/ 2 when the DOS box is about to be placed in the background and the video driver should save any necessary state.
**See also:** Function 4002h, MS Windows Function 4005h (Chapter 43)

---

**INT 2Fh Function 4002h** *OS/2 compatibility box Callout*

***SWITCHING DOS TO FOREGROUND***

**Registers at call:**
AX = 4002h

**Return Registers:**
none

**Details:** This function is called by OS/ 2 when the DOS box is about to be placed in the foreground and the video driver should restore the previously-saved state.
**See also:** Function 4001h, MS Windows Function 4006h (Chapter 43)

---

**INT 50h to INT 57h** *OS/2 v1.x*

***RELOCATED IRQ0 through IRQ7***

reprograms the interrupt controller to locate the hardware interrupts at different vectors for the dual purpose of easing the distinction between CPU-generated interrupts and external hardware interrupts, and being able to trap such important interrupts as the keyboard interrupt even if some application entirely takes over the usual interrupt vector.

## Miscellaneous

**INT 2Ch** *STARLITE architecture*

***KERNEL API***

**Details:** STARLITE is an architecture by General Software for a series of MS-DOS compatible operating systems (OEM DOS, NETWORK DOS, and SMP DOS) which were to be released in 1991. The interrupt number is subject to change before the actual release. It appears that the only operating system based on this architecture actually to be release is General Software's Embedded DOS.

## CHAPTER 27

# *MS-DOS Utilities*

This chapter describes 12 of the major utilities that are (or have recently been) bundled into MS-DOS, and which many users consider to be an integral part of the operating system. The 12 include: ANSI.SYS, APPEND, ASSIGN, the Critical Error Handler, DISPLAY.SYS, GRAFTABL, GRAPHICS, KEYB, NLSFUNC, PRINT, SHARE, and XMA2EMS. Finally, a "Miscellaneous" section wraps up loose ends.

In addition to the utilities listed in this chapter, several MS-DOS programs are covered in other chapters: COMMAND.COM in Chapter 28, DOSSHELL and SHELLB in Chapter 49, and EGA.SYS in Chapter 43.

## ANSI.SYS

The ANSI.SYS device driver replaces the standard console (CON) device driver with a new one which is capable of responding to a subset of the ANSI X3.64 terminal control sequences.

---

### INT 2Fh Function 1A00h

*DOS 4+ ANSI.SYS*

***INSTALLATION CHECK***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1A00h | AL = FFh if installed |

**Details:** AVATAR.SYS also responds to this call.

This function is documented for DOS 5+, but is undocumented for DOS 4.x.

---

### INT 2Fh Function 1A01h

*DOS 4+ ANSI.SYS internal (undocumented)*

***GET/SET DISPLAY INFORMATION***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1A01h | CF set on error |
| CL = function | AX = error code (many non-standard) |
| 7Fh for GET | CF clear if successful |
| 5Fh for SET | AX destroyed |
| DS:DX -> parameter block as for INT 21h Function 440Ch Subfunctions 037Fh/035Fh respectively | |

**Details:** Presumably this is the DOS IOCTL interface to ANSI.SYS.

**See also:** Function 1A02h, MS-DOS INT 21h Function 440Ch (Chapter 24)

---

### INT 2Fh Function 1A02h

*DOS 4+ ANSI.SYS internal (undocumented)*

***MISCELLANEOUS REQUESTS***

**Registers at call:**

AX = 1A02h

DS:DX -> parameter block
(see Table 27-1)

**Return Registers:**
none

**Details:** DOS 5+ chains to the previous handler if AL > 02h on call.
**See also:** Function 1A01h

**27-1 Format of ANSI.SYS parameter block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | subfunction<br>00h set/reset interlock<br>01h get /L flag |
| 01h | BYTE | interlock state<br>00h=reset, 01h=set<br>This interlock prevents some of the ANSI.SYS post-processing in its hook onto INT 10h Function 00h mode set |
| 02h | BYTE | (returned)<br>00h if /L not in effect<br>01h if /L in effect |

## APPEND

APPEND allows the user to specify a list of directories in which to search for files that are not found in the current directory, and transparently substitute a file from one of those directories for the one actually specified by the application on opening the file.

---

### INT 2Fh Function B700h *INSTALLATION CHECK*

*APPEND*

**Registers at call:**
AX = B700h

**Return Registers:**
AL = status
00h not installed
FFh installed

**Details:** MS-DOS 3.30 APPEND refuses to install itself when run inside TopView or a TopView-compatible environment.

---

### INT 2Fh Function B701h *GET APPEND PATH*

*APPEND v3.21 only (undocumented)*

**Registers at call:**
AX = B701h

**Return Registers:**
ES:DI -> active APPEND path

**Details:** The only version of APPEND known to support this call is the APPEND shipped with Microtek MS-DOS 3.21; MS-DOS 3.30-6.00 APPEND displays "Incorrect APPEND Version" and aborts the caller.

Use Function B704h first, and only call this function if that one is not supported.

**See also:** Function B704h

---

### INT 2Fh Function B702h *VERSION CHECK*

*APPEND*

**Registers at call:**
AX = B702h

**Return Registers:**
AX = FFFFh if not DOS 4.0 APPEND (also if DOS 5.0 APPEND)
otherwise,
AL = major version number
AH = minor version number

**See also:** Functions B700h and B710h

---

## INT 2Fh Function B703h

*DOS 3.3, DOS 5.0 APPEND (undocumented)*

### *HOOK INT 21h*

**Registers at call:**
AX = B703h
ES:DI -> INT 21h handler APPEND should chain to

**Return Registers:**
ES:DI -> APPEND's INT 21h handler

**Details:** Each invocation of this function toggles a flag which APPEND uses to determine whether to chain to the user handler or the original INT 21h.

---

## INT 2Fh Function B704h

*DOS 3.3+ APPEND*

### *GET APPEND PATH*

**Registers at call:**
AX = B704h

**Return Registers:**
ES:DI -> active APPEND path (128 bytes max)

**Details:** Some versions of append do not support this call, and return ES unchanged; in this case, you should call Function B701h to get the APPEND path.

**See also:** Function B701h

---

## INT 2Fh Function B706h

*DOS 4+ APPEND*

### *GET APPEND FUNCTION STATE*

**Registers at call:**
AX = B706h

**Return Registers:**
BX = APPEND state (see Table 27-2)

**See also:** Function B707h

**27-2 Bitfields for APPEND state:**

| | |
|---|---|
| bit 0 | APPEND enabled |
| bits 1-11 | reserved |
| bit 12 | (DOS 5.0) APPEND applies directory search even if a drive has been specified |
| bit 13 | /PATH flag active |
| bit 14 | /E flag active (environment var APPEND exists) |
| bit 15 | /X flag active |

---

## INT 2Fh Function B707h

*DOS 4+ APPEND*

### *SET APPEND FUNCTION STATE*

**Registers at call:**
AX = B707h
BX = APPEND state bits (see Table 27-2)

**Return Registers:**
none

**See also:** Function B706h

---

## INT 2Fh Function B710h

*DOS 3.3+ APPEND (undocumented)*

### *GET VERSION INFO*

**Registers at call:**
AX = B710h

**Return Registers:**
AX = current APPEND state (see Table 27-2)
BX = *unknown* (0000h in MS-DOS 3.30 and 5.00)
CX = *unknown* (0000h in MS-DOS 3.30 and 5.00)
DL = major version
DH = minor version

**See also:** Function B702h

---

## INT 2Fh Function B711h

*DOS 4+ APPEND*

### *SET RETURN FOUND NAME STATE*

**Registers at call:**
AX = B711h

**Return Registers:**
none

**Details:** If the next INT 21h call (and *only* the next) is function 3Dh, 43h, or 6Ch (also 4B03h and 4Eh if /X is active), the fully qualified filename is written over top of the filename passed to the INT 21h call. The application must provide a sufficiently large buffer. This state is reset after the next INT 21h call processed by APPEND.
**BUG:** DOS 4.0 APPEND reportedly overwrites DS:DX instead of DS:SI for INT 21h Function 6Ch.
**See also:** INT 21h Function 4Eh

## ASSIGN

ASSIGN is a small resident utility which redirects accesses from one drive to another. Although SUBST now provides most of the functionality of ASSIGN without any resident code, SUBST is not able to swap two drives like ASSIGN can (with the command *ASSIGN X=Y Y=X*).

---

### INT 2Fh Function 0600h
*DOS 3+ ASSIGN*

***INSTALLATION CHECK***

**Registers at call:**
AX = 0600h

**Return Registers:**
AL = status
  00h not installed
  01h not installed, but not OK to install
  FFh installed

**Details:** ASSIGN is not a TSR in DR-DOS 5.0; it is internally replaced by SUBST (see Table 24-82).

This function was undocumented prior to the release of DOS 5.0.

**See also:** Function 0601h, MS-DOS INT 21h Function 52h (Chapter 24)

---

### INT 2Fh Function 0601h
*DOS 3+ ASSIGN (undocumented)*

***GET DRIVE ASSIGNMENT TABLE***

**Registers at call:**
AX = 0601h

**Return Registers:**
ES = segment of ASSIGN work area and assignment table

**Details:** Under DOS 3+, the 26 bytes starting at ES:0103h specify to which drive each of A: to Z: is mapped. This table is initially set to 01h 02h 03h....
**See also:** Function 0600h

## Critical Error Handler

As of DOS 3.0, COMMAND.COM provides a hook for other programs to override the default error messages for INT 24h, and to provide messages for error numbers not understood by COMMAND.COM.This hook is used by network redirectors and MSCDEX to provide error messages for network or CD-ROM errors.

---

### INT 2Fh Function 0500h
*DOS 3+ CRITICAL ERROR HANDLER (undocumented)*

***INSTALLATION CHECK***

**Purpose:** Determine whether a critical error message override is installed.

**Registers at call:**
AX = 0500h

**Return Registers:**
AL = status
  00h not installed, OK to install
  01h not installed, can't install
  FFh installed

**See also:** Function 05h below, INT 24h (Chapter 24)

### INT 2Fh Function 05h
***EXPAND ERROR INTO STRING***

*DOS 3+ CRITICAL ERROR HANDLER*
*Callout (undocumented)*

**Registers at call:**
AH = 05h
**---DOS 3.x---**
AL = extended error code (nonzero)
**---DOS 4+ ---**
AL = error type
01h DOS extended error code
02h parameter error
BX = error code

**Return Registers:**
CF clear if successful
ES:DI -> ASCIZ error message (read-only)
AL = completion state
00h message requires completion with device name, drive, etc.
01h message is complete as returned
CF set if error code can't be converted to string
AX,DI,ES destroyed
other flags corrupted

**Details:** This function is called at the start of COMMAND.COM's default critical error handler if installed by a user program, allowing partial or complete overriding of the default error messages.

Subfunction 02h is called by many DOS 4 external programs.

DR-DOS's COMMAND.COM appends additional information ("0 files copied") to the returned string.

**See also:** Function 0500h, MS-DOS kernel Function 122Eh (Chapter 24), INT 24h (Chapter 24)

## DISPLAY.SYS

DISPLAY.SYS is the device driver supplied in MS-DOS versions 3.3 and above to support international localization of the CRT display.

### INT 2Fh Function AD00h
***INSTALLATION CHECK***

*DOS 3.3+ DISPLAY.SYS internal*
*(undocumented)*

**Registers at call:**
AX = AD00h

**Return Registers:**
AL = FFh if installed
BX = *unknown* (0100h for MS-DOS 3.3+)

**Details:** DOS 5+ DISPLAY.SYS chains to the previous handler if AL is not one of the subfunctions listed here.

### INT 2Fh Function AD01h
***SET ACTIVE CODE PAGE***

*DOS 3.3+ DISPLAY.SYS internal*
*(undocumented)*

**Registers at call:**
AX = AD01h
BX = new code page

**Return Registers:**
CF clear if successful
AX = 0001h
CF set on error (unsupported code page)
AX = 0000h

**See also:** Function AD02h

### INT 2Fh Function AD02h
***GET ACTIVE CODE PAGE***

*DOS 3.3+ DISPLAY.SYS internal*
*(undocumented)*

**Registers at call:**
AX = AD02h

**Return Registers:**
CF set if code page never set
AX = 0001h
BX = FFFFh (assume first hardware code page)
CF clear if successful
BX = current code page

**See also:** Functions AD01h and AD03h

### INT 2Fh Function AD03h
### *GET CODE PAGE INFORMATION*

*DOS 3.3+ DISPLAY.SYS internal (undocumented)*

**Registers at call:**
AX = AD03h
ES:DI -> buffer for code page information (see Table 27-3)
CX = size of buffer in bytes

**Return Registers:**
CF set if buffer too small
CF clear if successful
ES:DI buffer filled

**See also:** Functions AD01h and AD02h

**27-3 Format of DOS 5.0-6.0 code page information:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of software code pages |
| 02h | WORD | *unknown* (0003h) |
| 04h | WORD | number of hardware code pages |
| 06h | N WORDs | hardware code page numbers |
|  | N WORDs | software (prepared) code pages (FFFFh if not yet prepared) |

### INT 2Fh Function AD04h
### *UNDETERMINED FUNCTION*

*DOS 4.x only DISPLAY.SYS internal (undocumented)*

**Registers at call:**
AX = AD04h
*others, if any, unknown*

**Return Registers:**
*unknown*

### INT 2Fh Function AD10h
### *INSTALLATION CHECK*

*DOS 4.x DISPLAY.SYS internal (undocumented)*

**Registers at call:**
AX = AD10h
*others, if any, unknown*

**Return Registers:**
AX = FFFFh
BX = *unknown* (0100h in PC-DOS 4.01)

### INT 2Fh Function AD10h
### *UNDETERMINED FUNCTION*

*DOS 5+ DISPLAY.SYS internal (undocumented)*

**Registers at call:**
AX = AD10h
*others, if any, unknown*

**Return Registers:**
CF clear if successful
CF set on error

**Details:** This function is a NOP if the active code page has never been set (Function AD02h returns BX=FFFFh); its purpose otherwise is not known.

## GRAFTABL

GRAFTABL is a small TSR which provides graphical character fonts for the CGA, which does not support hardware fonts like the EGA or VGA.

### INT 2Fh Function B000h
### *INSTALLATION CHECK*

*DOS 3.3+ GRAFTABL.COM*

**Registers at call:**
AX = B000h

**Return Registers:**
AL = status
00h not installed, OK to install
01h not installed, not OK to install
FFh installed

**Details:** This function is called by DISPLAY.SYS. It is documented for DOS 5.0, but was undocumented in prior versions.
**Conflicts:** none, but see Table 2-2
**See also:** Function B001h

---

**INT 2Fh Function B001h** *DOS 3.3+ GRAFTABL.COM*

***GET GRAPHICS FONT TABLE***

**Registers at call:**
AX = B001h
DS:BX -> DWORD buffer for address of 8x8 font table

**Return Registers:**
buffer filled
AL = FFh

**Details:** PC-DOS 3.30/4.01 and MS-DOS 6.0 set the font table offset to 0130h, MS-DOS 3.30 sets it to 0030h.

**See also:** Function B000h

## GRAPHICS

GRAPHICS is a TSR which provides PrintScreen support for graphics screens in addition to the ROM BIOS's support for text screens.

---

**INT 2Fh Function 1500h** *DOS 4.00 only GRAPHICS.COM*

***INSTALLATION CHECK***

**Registers at call:**
AX = 1500h

**Return Registers:**
AX = FFFFh
ES:DI -> *unknown* (graphics data?)

**Details:** This installation check conflicts with the CD-ROM Extensions installation checks; it was moved to Function AC00h in later versions.

**See also:** Function AC00h

---

**INT 2Fh Function AC00h** *DOS 4.01+ GRAPHICS.COM*

***INSTALLATION CHECK***

**Registers at call:**
AX = AC00h

**Return Registers:**
AX = FFFFh
ES:DI -> *unknown* (graphics data?) (not documented)

**Details:** This installation check was moved here to avoid the conflict with the CD-ROM extensions that occurred in DOS 4.00.

**See also:** Function 1500h

## KEYB

KEYB is a generic keyboard redefinition TSR for National Language Support. It replaces the language-specific keyboard redefinition programs KEBYxx.COM provided in earlier DOS versions.

---

**INT 2Fh Function AD80h** *DOS 3.3+ KEYB.COM internal (partially doc.)*

***INSTALLATION CHECK***

**Registers at call:**
AX = AD80h

**Return Registers:**
AL = FFh if installed
BX = version number (BH = major, BL = minor)
ES:DI -> internal data (see Table 27-4)

**Details:** MS-DOS 3.30, PC-DOS 4.01, and MS-DOS 5.00-6.00 all report version 1.00.

This function was undocumented prior to the release of DOS 5.0.

**See also:** Functions AD81h and AD82h

**27-4 Format of KEYB internal data:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | original INT 09h |
| 04h | DWORD | original INT 2Fh |

| | | |
|---|---|---|
| 08h | 6 BYTEs | *unknown* |
| 0Eh | WORD | flags |
| 10h | BYTE | *unknown* |
| 11h | BYTE | *unknown* |
| 12h | 4 BYTEs | *unknown* |
| 16h | 2 BYTEs | country ID letters |
| 18h | WORD | current code page |
| **---DOS 3.3---** | | |
| 1Ah | WORD | pointer to *first item in list of code page tables* |
| 1Ch | WORD | pointer to *unknown* item in list of code page tables |
| 1Eh | 2 BYTEs | *unknown* |
| 20h | WORD | pointer to key translation data |
| 22h | WORD | pointer to last item in code page table list (see Table 27-5) |
| 24h | 9 BYTEs | *unknown* |
| **---DOS 4.01---** | | |
| 1Ah | 2 BYTEs | *unknown* |
| 1Ch | WORD | pointer to *first item in list of code page tables* |
| 1Eh | WORD | pointer to *unknown* item in list of code page tables |
| 20h 2 | BYTEs | *unknown* |
| 22h | WORD | pointer to key translation data |
| 24h | WORD | pointer to last item in code page table list (see Table 27-5) |
| 26h | 9 BYTEs | *unknown* |

**27-5 Format of code page table list entries:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | pointer to next item, FFFFh = last |
| 02h | WORD | code page |
| 04h | 2 BYTEs | *unknown* |

**27-6 Format of translation data:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | size of data in bytes, including this word |
| 02h | N-2 BYTEs | *unknown* |

## INT 2Fh Function AD81h

*DOS 3.3+ KEYB.COM*

### *SET KEYBOARD CODE PAGE*

**Registers at call:**
AX = AD81h
BX = code page (see Table 24-119)

**Return Registers:**
CF set on error
AX = 0001h (code page not available)
CF clear if successful

**Details:** This function is called by DISPLAY.SYS. It was undocumented prior to the release of DOS 5.0.

**See also:** Function AD82h

## INT 2Fh Function AD82h

*DOS 3.3+ KEYB.COM*

### *SET KEYBOARD MAPPING*

**Registers at call:**
AX = AD82h
BL = new state
00h US keyboard (Control-Alt-F1)
FFh foreign keyboard (Control-Alt-F2)

**Return Registers:**
CF set on error (BL not 00h or FFh)
CF clear if successful

**Details:** This function was undocumented prior to the release of DOS 5.0.

**See also:** Functions AD81h and AD83h

### INT 2Fh Function AD83h

*DOS 5+ KEYB.COM*

### *GET KEYBOARD MAPPING*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = AD83h | BL = current state<br>00h US keyboard<br>FFh foreign keyboard |

**See also:** Function AD82h

## NLSFUNC

NLSFUNC provides National Language Support in the form of implementing the DOS country information functionality for countries other than the default country configured into DOS.

### INT 2Fh Function 1400h

*NLSFUNC.COM*

*Callout*

### *INSTALLATION CHECK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1400h | AL = status<br>00h not installed, OK to install<br>01h not installed, not OK<br>FFh installed |

**Details:** This function is called by the DOS v3.3+ kernel. It is documented for MS-DOS 5+, but was undocumented in prior versions.

This function is also supported by the OS/2 v1.3+ compatibility box, which always returns AL=FFh, and by DR-DOS 5.0.

**Conflicts:** European MS-DOS 4.0 POPUP (Chapter 25)

**See also:** Functions 1401h and 1402h

### INT 2Fh Function 1401h

*NLSFUNC.COM*

*Callout (undocumented)*

### *CHANGE CODE PAGE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1401h<br>DS:SI -> internal code page structure (see Table 27-7)<br>BX = new code page (see Table 24-119)<br>DX = *country code* | AL = status<br>00h successful<br>else DOS error code (see Table 24-2) |

**Details:** This function is called by the DOS v3.3+ kernel.

**Conflicts:** European MS-DOS 4.0 POPUP (Chapter 25)

**See also:** Functions 1400h and 1402h, MS-DOS INT 21h Function 66h (Chapter 24)

**27-7 Format of DOS 3.30 internal code page structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 8 BYTEs | *unknown* |
| 08h | 64 BYTEs | name of country information file |
| 48h | WORD | system code page |
| 4Ah | WORD | number of supported subfunctions |
| 4Ch | 5 BYTEs | data to return for INT 21h Function 6502h |
| 51h | 5 BYTEs | data to return for INT 21h Function 6504h |
| 56h | 5 BYTEs | data to return for INT 21h Function 6505h |
| 5Bh | 5 BYTEs | data to return for INT 21h Function 6506h |
| 60h | 41 BYTEs | data to return for INT 21h Function 6501h |

### INT 2Fh Function 1402h
### *GET EXTENDED COUNTRY INFORMATION*

*NLSFUNC.COM*
*Callout (undocumented)*

**Registers at call:**
AX = 1402h
BP = subfunction (same as AL for INT 21h Function 65h)
BX = code page (see Table 24-119)
DX = country code (see Table 24-18)
DS:SI -> internal code page structure (see Table 27-7)
ES:DI -> user buffer
CX = size of user buffer

**Return Registers:**
AL = status
- 00h successful
- else DOS error code (see Table 24-2)

**Details:** This function is called by the DOS v3.3+ kernel on INT 21h Function 65h.

The code page structure is apparently only needed for the COUNTRY.SYS pathname.

**Conflicts:** European MS-DOS 4.0 POPUP (Chapter 25)

**See also:** Functions 1401h, 1403h, and 1404h; MS-DOS INT 21h Function 65h (Chapter 24)

### INT 2Fh Function 1403h
### *SET CODE PAGE*

*NLSFUNC.COM*
*Callout (undocumented)*

**Registers at call:**
AX = 1403h
DS:SI -> internal code page structure (see Table 27-7)
BX = code page (see Table 24-119)
DX = country code (see Table 24-18)

**Return Registers:**
AL = status
*unknown*

**Details:** This function is called by the DOS v3.3+ kernel on INT 21h Function 38h.

**Conflicts:** European MS-DOS 4.0 POPUP (Chapter 25)

**See also:** Functions 1402h and 1404h, MS-DOS INT 21h Function 38h (Chapter 24)

### INT 2Fh Function 1404h
### *GET COUNTRY INFORMATION*

*NLSFUNC.COM*
*Callout (undocumented)*

**Registers at call:**
AX = 1404h
BX = code page (see Table 24-119)
DX = country code (see Table 24-18)
DS:SI -> internal code page structure (see Table 27-7)
ES:DI -> user buffer

**Return Registers:**
AL = status
*unknown*

**Details:** This function is called by the DOS v3.3+ kernel on INT 21h Function 38h.

The code page structure is apparently only needed for the COUNTRY.SYS pathname.

**Conflicts:** European MS-DOS 4.0 POPUP (Chapter 25)

**See also:** Functions 1402h and 1403h, MS-DOS INT 21h Function 38h (Chapter 24)

## SHARE

SHARE provides file sharing protections to prevent a program from improperly accessing a file which has already been opened by another program on the network or under a multitasker. It also provides the implementation of the record-locking calls supported by DOS.

### INT 21h Function 5D02h
### *SHARE.EXE - CLOSE FILE BY NAME*

*DOS 3.1+ internal*
*(undocumented)*

**Registers at call:**
AX = 5D02h
DS:DX -> DOS parameter list

(see Table 24-107), only fields DX, DS, computer ID, and process ID used
DPL's DS:DX -> ASCIZ name of file to close

**Return Registers:**
CF set on error
AX = error code (see Table 24-2)
CF clear if successful

**Details:** This function can only succeed if SHARE is loaded, because it calls [SysFileTable-28h] (see Table 24-76).

The filename must be canonical and fully-qualified, such as returned by Function 60h (Chapter 24).

This function is not supported by DR-DOS 3.41 and 5.0; it returns error code 01h.

**See also:** Functions 5D03h and 5D04h, MS-DOS Functions 3Eh and 60h (Chapter 24)

---

### INT 21h Function 5D03h
### *SHARE.EXE - CLOSE ALL FILES FOR GIVEN COMPUTER*

*DOS 3.1+ internal (undocumented)*

**Registers at call:**
AX = 5D03h
DS:DX -> DOS parameter list (see Table 24-107), only computer ID used

**Return Registers:**
CF set on error
AX = error code (see Table 24-2)
CF clear if successful

**Details:** This function can only succeed if SHARE is loaded, because it calls [SysFileTable-30h] (see Table 24-76).

This function is not supported by DR-DOS 3.41 and 5.0; it returns error code 01h.

**See also:** Functions 5D02h and 5D04h

---

### INT 21h Function 5D04h
### *SHARE.EXE - CLOSE ALL FILES FOR GIVEN PROCESS*

*DOS 3.1+ internal (undocumented)*

**Registers at call:**
AX = 5D04h
DS:DX -> DOS parameter list (see Table 24-107), only computer ID and process ID fields used

**Return Registers:**
CF set on error
AX = error code (see Table 24-2)
CF clear if successful

**Details:** This function can only succeed if SHARE is loaded, because it calls [SysFileTable-2Ch] (see Table 24-76).

This function is not supported by DR-DOS 3.41 and 5.0; it returns error code 01h.

**See also:** Functions 5D02h and 5D03h, Network Redirector INT 2Fh Function 111Dh (NI)

---

### INT 21h Function 5D05h
### *SHARE.EXE - GET OPEN FILE LIST ENTRY*

*DOS 3.1+ internal (undocumented)*

**Registers at call:**
AX = 5D05h
DS:DX -> DOS parameter list (see Table 24-107) DPL's BX = index of sharing record (see Table 24-77)
DPL's CX = index of SFT in sharing record's SFT list

**Return Registers:**
CF clear if successful
ES:DI -> ASCIZ filename
BX = network machine number of SFT's owner
CX = number of locks held by SFT's owner
CF set if either index out of range
AX = 0012h (no more files)

**Details:** This function can only succeed if SHARE is loaded, because it calls [SysFileTable-18h] (see Table 24-76).

The filenames are always canonical and fully-qualified, such as returned by Function 60h (Chapter 24).

This function is not supported by DR-DOS 3.41 and 5.0, but does not return an error.

**See also:** MS-DOS Functions 5Ch and 60h (Chapter 24)

### INT 2Fh Function 1000h

*SHARE*

### *INSTALLATION CHECK*

**Registers at call:**
AX = 1000h

**Return Registers:**
AL = status
00h not installed, OK to install
01h not installed, not OK to install
FFh installed

**BUGS:** Values of AL other than 00h put DOS 3.x SHARE into an infinite loop; values of AL other than described here put PC-DOS 4.00 into the same loop (the buggy instructions are the same).

**Details:** This function is supported by the OS/2 v1.3+ compatibility box, which always returns AL=FFh.

If DOS 4.01 SHARE was automatically loaded, file sharing is in an inactive state (due to the undocumented /NC flag used by the autoload code) until this call is made.

DOS 5+ chains to the previous handler if AL was nonzero on entry.

Windows Enhanced mode hooks this call and reports that SHARE is installed even when it is not.

**See also:** Function 1080h, MS-DOS INT 21h Function 52h (Chapter 24)

### INT 2Fh Function 1040h

*DOS 4 only SHARE internal (undocumented)*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 1040h
*others, if any, unknown*

**Return Registers:**
*AL = FFh*

**See also:** Function 1000h

### INT 2Fh Function 1080h

*DOS 4 only SHARE internal (undocumented)*

### *TURN ON FILE SHARING CHECKS*

**Registers at call:**
AX = 1080h

**Return Registers:**
AL = status
F0h successful
FFh checking was already on

**Details:** DOS 4.x SHARE has dual functions: FCB support for large (>32M) media and file sharing checks. The undocumented commandline flag /NC can be used to disable the sharing code.

**See also:** Functions 1000h and 1081h

### INT 2Fh Function 1081h

*DOS 4 only SHARE internal (undocumented)*

### *TURN OFF FILE SHARING CHECKS*

**Registers at call:**
AX = 1081h

**Return Registers:**
AL = status
F0h successful
FFh checking was already off

**Details:** (see Function 1080h)

**See also:** Function 1000h and 1080h

## XMA2EMS

The XMA2EMS.SYS extension is installed only if DOS has page frames to hide. This extension hooks onto INT 67h Function 58h and returns data from that call which excludes the physical pages being used by DOS.

---

**INT 2Fh Function 1B00h** *DOS 4+ XMA2EMS.SYS internal (undocumented)*

***INSTALLATION CHECK***

**Registers at call:**
AX = 1B00h

**Return Registers:**
AL = FFh if installed

**See also:** Function 1Bh below

---

**INT 2Fh Function 1Bh** *DOS 4+ XMA2EMS.SYS internal (undocumented)*

***GET HIDDEN FRAME INFORMATION***

**Registers at call:**
AH = 1Bh
AL nonzero
DI = hidden physical page number

**Return Registers:**
AX = FFFFh if failed (no such hidden page)
AX = 0000h if OK, then
  ES = segment of page frame
  DI = physical page number

**Details:** The returned information corresponds to the data edited out of the INT 67h Function 58h call.

FASTOPEN makes this call with AL = FFh.

**See also:** Function 1B00h

## Miscellaneous

---

**INT 16h Function 92h** *UNKNOWN*

***UNDETERMINED FUNCTION***

**Registers at call:**
AH = 92h

**Return Registers:**
AH <= 80h if *unknown*

**Details:** Called by DOS 3.2 KEYBxx.COM and DOS 5+ KEYB.COM.

**See also:** Function A2h, PCjr Function 05h (Chapter 4)

---

**INT 16h Function A2h** *UNKNOWN*

***UNDETERMINED FUNCTION***

**Registers at call:**
AH = A2h

**Return Registers:**
AH <= 80h if *unknown*

**Details:** This function is called by DOS 5+ KEYB.COM.

**See also:** Function 92h

---

**INT 2Fh Function 49h** *UNKNOWN (undocumented)*

***UNDETERMINED FUNCTION***

**Registers at call:**
AH = 49h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Reportedly called by the DOS 5.0 installation program.

---

**INT 2Fh Function 4A06h** *DOS 5+ Callout (undocumented)*

***DOS SUPERVISOR "REBOOT PANEL" - ADJUST MEMORY SIZE***

**Purpose:** This function is used to override the default memory size when booting diskless workstations.

**Registers at call:**
AX = 4A06h
DX = segment following last byte of conventional memory

**Return Registers:**
DX = segment following last byte of memory available for use by DOS

**Details:** The MS-DOS 5+ IO.SYS startup code calls this function if the signature "RPL" is present three bytes beyond the INT 2Fh handler; the result of this call overrides the value returned by INT 12h.

This function is normally hooked by RPL code at the top of memory to protect itself from being overwritten; DOS builds a memory block with owner = 0008h and name "RPL" which must be freed by the RPL code when it is done.
**See also:** INT 12h (Chapter 4)

---

### INT 2Fh Function 4A07h — *RESERVED FOR PROTMAN SUPPORT*
### *UNDETERMINED FUNCTION* — *(undocumented)*

**Registers at call:**
AX = 4A07h
*others, if any, unknown*

**Return Registers:**
*unknown*

---

### INT 2Fh Function AD40h — *DOS 4+*
### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = AD40h
DX = *unknown*
*others, if any, unknown*

**Return Registers**

**Details:** This function is called by PC-DOS 4.01 PRINT.COM.

# CHAPTER 28

# *COMMAND.COM and Replacements*

Although most people think of COMMAND.COM as "DOS", it is really "just another program" which can easily be replaced with the SHELL= directive in CONFIG.SYS. In fact, other than dealing with several items (environment, Ctrl-Break interrupt, and critical error interrupt) that are not yet fully initialized at the time it is run, there is little difference between the selected command interpreter and other applications. This chapter covers API functions provided by COMMAND.COM and its major replacement, 4DOS and the licensed variant thereof NDOS.

## COMMAND.COM

### INT 2Eh *DOS 2+*

### *PASS COMMAND TO COMMAND INTERPRETER FOR EXECUTION* *(undocumented)*

**Purpose:** This call allows execution of arbitrary commands (including COMMAND.COM internal commands) without loading another copy of COMMAND.COM.

| **Registers at call:** | **Return Registers:** |
|---|---|
| DS:SI -> commandline to execute<br>(see Table 28-1) | all registers except CS:IP destroyed<br>AX = status (4DOS v4.0)<br>0000h successful<br>FFFFh error before processing command (not enough memory, etc)<br>other error number returned by command |

**Details:** If COMMAND.COM is the user's command interpreter, the primary copy executes the command; this allows the master environment to be modified by issuing a "SET" command, but changes in the master environment will not become effective until all programs descended from the primary COMMAND.COM terminate. Since COMMAND.COM processes the string as if typed from the keyboard, the transient portion needs to be present, and the calling program must ensure that sufficient memory to load the transient portion can be allocated by DOS if necessary.

The results are unpredictable if invoked by a program run from a batch file because this call is not reentrant and COMMAND.COM uses the same internal variables when processing a batch file.

This interrupt is hooked but ignored by the 4DOS v3.0 COMMAND.COM replacement unless SHELL2E has been loaded. Version 4 provides a configuration parameter to specify whether 4DOS should only hook the interrupt or actually honor requests.

The MS-DOS 5 Programmer's Reference calls this interrupt "Reload Transient".

**28-1 Format of commandline:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | length of command string, not counting trailing CR |
| 01h | var | command string |
|  | N BYTE | 0Dh (CR) |

### INT 2Fh Function 5500h
### *COMMAND.COM INTERFACE*

*DOS 5+*
*(undocumented)*

**Purpose:** This function is used to access the shareable portion of COMMAND.COM, which may have been moved into the HMA; only the primary COMMAND.COM retains this portion.

| Registers at call: | Return Registers: |
|---|---|
| AX = 5500h | AX = 0000h if an instance of COMMAND.COM is already running<br>DS:SI -> entry point table |

**Details:** Procedures are called from a dispatcher in COMMAND's resident portion; most assume that the segment address of the resident portion is on the stack and are thus not of general use.

**See also:** Function 5501h

### INT 2Fh Function 5501h
### *ROM COMMAND.COM INTERFACE*

*DOS 5+*
*(undocumented)*

**Purpose:** This function is used to determine whether the caller is the first instance of ROM COMMAND.COM.

| Registers at call: | Return Registers: |
|---|---|
| AX = 5501h | *unknown* |

**See also:** Function 5500h

### INT 2Fh Function AE00h
### *INSTALLABLE COMMAND - INSTALLATION CHECK*

*DOS 3.3+ COMMAND.COM*
*Callout (undocumented)*

**Purpose:** This call provides a mechanism for TSRs to install permanent extensions to the command repertoire of COMMAND.COM. COMMAND.COM makes this call before executing the current command line, and does not execute it itself if the return is FFh.

| Registers at call: | Return Registers: |
|---|---|
| AX = AE00h<br>DX = FFFFh<br>CH = FFh<br>CL = length of command line tail (4DOS v4.0)<br>DS:BX -> command line buffer (see below)<br>DS:SI -> command name buffer (see below)<br>DI = 0000h (4DOS v4.0) | AL = FFh if this command is a TSR extension to COMMAND.COM<br>AL = 00h if the command should be executed as usual |

**Details:** APPEND hooks this call, to allow subsequent APPEND commands to execute without re-running APPEND.

**See also:** Function AE01h

**Format of command line buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | max length of command line, as in INT 21/AH=0Ah |
| 01h | BYTE | count of bytes to follow, excluding terminating 0Dh |
| | N BYTEs | command line text, terminated by 0Dh |

**Format of command name buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | length of command name |
| 01h | N BYTEs | uppercased command name (blank-padded to 11 chars by 4DOS v4) |

### INT 2Fh Function AE01h
### *INSTALLABLE COMMAND - EXECUTE*

*DOS 3.3+ COMMAND.COM*
*Callout (undocumented)*

**Purpose:** This call requests execution of the command which a previous call to Function AE00h indicated was resident.

**Registers at call:**
AX = AE01h

| | **Return Registers:** |
|---|---|
| DX = FFFFh | DS:SI buffer updated |
| CH = 00h | |
| CL = length of command name (4DOS v4.0) | |
| DS:SI -> command name buffer (see AX=AE00h) | |
| DS:BX -> command line buffer (see AX=AE00h) | |

**Details:** If the length byte of the returned buffer is nonzero, the following bytes contain the uppercase internal command to execute and the command line buffer contains the command's parameters (the first DS:[SI] bytes are ignored).

APPEND hooks this call.

**See also:** Function AE00h

## 4DOS / NDOS

4DOS is a shareware COMMAND.COM replacement by Rex Conn and JP Software Inc. NDOS is a variant of 4DOS licensed to Symantec and included with The Norton Utilities.

### INT 2Eh *4DOS v2.x-3.03 SHELL2E.COM*

### *UNINSTALL*

| **Registers at call:** | **Return Registers:** |
|---|---|
| BX = E22Eh | if successful, SHELL2E terminates itself with INT 21h Function 4Ch |
| DS:SI -> zero byte | |

### INT 2Fh Function D44Dh *4DOS.COM v2.1+*

### *API*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = D44Dh | |
| BH = function | |
| 00h installation check | AX = 44DDh<br>BL = major version number<br>BH = minor version number<br>CX = PSP segment address for current invocation<br>DL = 4DOS shell number (0 for the first (root) shell, updated each time a new copy is loaded) |
| 01h (internal, v2.1-3.03) terminate current copy of 4DOS | nothing |
| (internal, v4.0) *unknown* | ES:BX -> data area (see Table 28-3) |
| 02h *unknown*<br>DX = *unknown* | |
| **---v2.1-3.03 only---** | |
| 03h EXEC program<br>CX:DX -> EXEC record | |
| FEh deallocate shell number (passed through to root shell)<br>*unknown* | |
| FFh allocate shell number (passed through to root shell) | |

**BUG:** Version 3.00 will crash the system if an unrecognized value in passed in BH.

**Conflicts:** none, but see Table 2-2

**See also:** Functions D44Eh and E44Dh, MS-DOS INT 21h Function 4403h (Chapter 24)

**28-2 Format of EXEC record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | offset of ASCIZ program name in same segment as EXEC record |
| 02h | WORD | offset of DOS commandline in same segment as EXEC record |
| 04h | WORD | segment of environment for child process (see Table 24-12) |

**28-3 Format of 4DOS v4.0 data area:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | *unknown* |
| 06h | WORD | XMS handle for swapping |
| | | *additional fields, if any, unknown* |

## INT 2Fh Function D44Eh
## *AWAITING USER INPUT*

*4DOS v3.0+*
*Callout*

**Registers at call:**
AX = D44Eh
---4DOS v3.01+---
BX = prompt status
    0000h 4DOS is ready to display prompt
    0001h 4DOS has displayed the prompt, about to accept user input

**Return Registers:**
handler must preserve SI, DI, BP, SP, DS, ES, and SS

**Details:** Version 3.00 only makes the call corresponding to BX=0001h, and does not set BX.

**Conflicts:** none, but see Table 2-2

**See also:** Function D44Dh

## INT 2Fh Function D44Fh, Subfunction 0000h
## *KSTACK.COM - INSTALLATION CHECK*

*4DOS v4.0+*

**Registers at call:**
AX = D44Fh
BX = 0000h

**Return Registers:**
AX = 44DDh if installed

**Conflicts:** none, but see Table 2-2

**See also:** Function D44Fh Subfunction 0001h

## INT 2Fh Function D44Fh, Subfunction 0001h
## *KSTACK.COM - PLACE KEYSTROKES INTO KEYSTACK*

*4DOS v4.0+*

**Registers at call:**
AX = D44Fh
BX = 0001h
CX = number of keystrokes (01h-FFh)
DS:DX -> keystroke list (one word per keystroke)

**Return Registers:**
AX = status
    0000h successful
    nonzero failed
BX,CX,DX destroyed

**Details:** The keystrokes are the exact values to return from subsequent calls to INT 16h Functions 00h, 01h, 10h, and 11h with the following exceptions:
    0000h causes subfunctions 01h and 11h to indicate an empty keyboard buffer
    FFFFh is followed by a word indicating the number of clock ticks to delay before the next faked keystroke.

The v4.00 KSTACK overwrites any unread keystrokes from the previous invocation, and does not range-check CX; it will overwrite memory following the resident portion if CX is greater than 100h.

**Conflicts:** none, but see Table 2-2

**See also:** Function D44Fh Subfunction 0000h, BIOS INT 16h Function 00h (Chapter 4), MS-DOS INT 21h Function 4403h (Chapter 24)

---

### INT 2Fh Function E44Dh

*NDOS*

***API***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = E44Dh | (see Function D44Dh) |

**Details:** As NDOS is a licensed version of 4DOS v3.03 and v4.0, the API is identical to that for 4DOS, except that AH is set to E4h instead of D4h and the installation check returns AX=44EEh instead of AX=44DDh.

**Conflicts:** none, but see Table 2-2

**See also:** Functions D44Dh,Function E44Eh

---

### INT 2Fh Function E44Eh

*NDOS*
*Callout*

***AWAITING USER INPUT***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = E44Eh<br>BX = prompt status<br>0000h NDOS is ready to display prompt<br>0001h NDOS has displayed the prompt, about to accept user input | handler must preserve SI, DI, BP, SP, DS, ES, and SS |

**Conflicts:** none, but see Table 2-2

**See also:** Function E44Dh

# CHAPTER 29

# *Virtual Control Program Interface (VCPI)*

VCPI was created out of the need to have multiple 386 protected-mode supervisors (such as a multitasker and a DOS extender) coexist. It provides rudimentary services which will allow a master supervisor program to stay in control of the system while still permitting other programs access to protected mode.

The VCPI specification contains only a small set of services; for a more complete set, see the DOS Protected-Mode Interface in Chapter 30.DPMI is unfortunately incompatible with VCPI.

---

**INT 67h Function DE00h** *VCPI*

***INSTALLATION CHECK***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE00h | AH = 00h VCPI is present<br>BH = major version number<br>BL = minor version number<br>AH nonzero VCPI not present |

**BUG:** MS Windows 3.00 is reported to "object violently" to this call.

**Conflicts:** see Table 2-3

**See also:** DPMI INT 2Fh Function 1687h (Chapter 30)

---

**INT 67h Function DE01h** *VCPI*

***GET PROTECTED MODE INTERFACE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE01h<br>ES:DI -> 4K page table buffer<br>DS:SI -> three descriptor table entries in GDT<br>(first becomes code segment descriptor, other two for use by main control program) | AH = 00h successful<br>DI -> first unused page table entry in buffer<br>EBX -> protected mode entry point in code segment<br>AH = nonzero failed |

**Details:** The protected mode entry point may be called with AX=DE00h-DE05h and AX=DE0Ch (in each case, all other registers as appropriate for the function).

**See also:** DPMI INT 2Fh Function 1687h (Chapter 30), QEMM INT 67h Function 3Fh (Table 39-18)

---

**29-1 QEMM v6.03 protected mode entry point calling convention:**

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DF00h *unknown*<br>*other registers, if any, unknown*<br>AX = DF01h *unknown*<br>*other registers, if any, unknown* | *unknown* |

### INT 67h Function DE02h

*VCPI*

### *GET MAXIMUM PHYSICAL MEMORY ADDRESS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE02h | AH = 00h successful<br>EDX = physical address of highest 4K memory page<br>AH nonzero: failed |

**See also:** QEMM Function 3Fh (Table 39-25)

### INT 67h Function DE03h

*VCPI*

### *GET NUMBER OF FREE 4K PAGES*

**Purpose:** Get total number of pages available to *all* tasks in the system.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE03h | AH = 00h successful<br>EDX = number of free 4K pages<br>AH nonzero: failed |

**Details:** This function is also available in protected mode by calling the protected-mode VCPI entry point.

**See also:** Function DE04h

### INT 67h Function DE04h

*VCPI*

### *ALLOCATE A 4K PAGE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE04h | AH = 00h if successful<br>EDX = physical address of allocated page<br>AH nonzero on failure |

**Details:** The client program is responsible for freeing all memory allocated with this call before terminating. This function is also available in protected mode by calling the protected-mode VCPI entry point.

**See also:** Functions DE03h and DE05h, QEMM Function 3Fh (Table 39-22)

### INT 67h Function DE05h

*VCPI*

### *FREE 4K PAGE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE05h<br>EDX = physical address of 4K page | AH = 00h if successful<br>AH nonzero on failure |

**Details:** This function is also available in protected mode by calling the protected-mode VCPI entry point.

**See also:** Function DE04h, QEMM Function 3Fh (Table 39-21)

### INT 67h Function DE06h

*VCPI*

### *GET PHYSICAL ADDRESS OF PAGE IN FIRST MEGABYTE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE06h<br>CX = page number (linear address shifted right 12 bits) | AH = 00h successful<br>EDX = physical address of page<br>AH nonzero: invalid page number (AH = 8Bh recommended) |

**See also:** QEMM Function 3Fh (Table 39-26), RM386 Function 5BE0h (Chapter 41)

### INT 67h Function DE07h

*VCPI*

### *READ CR0*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE07h | AH = 00h<br>EBX = value of Control Register 0 |

**See also:** Function DE07h, QEMM Function 3Fh (Table 39-21)

---

### INT 67h Function DE08h

*VCPI*

#### *READ DEBUG REGISTERS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE08h | AH = 00h |
| ES:DI -> array of 8 DWORDs | buffer filled with DR0 first, DR7 last, DR4 and DR5 unused |

**See also:** Function DE09h, QEMM Function 3Fh (Table 39-19)

---

### INT 67h Function DE09h

*VCPI*

#### *SET DEBUG REGISTERS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE09h | AH = 00h |
| ES:DI -> array of 8 DWORDs<br>holding new values of debug registers | |

**Details:** Any values for DR4 and DR5 are ignored.

**See also:** Function DE08h, QEMM Function 3Fh (Table 39-20)

---

### INT 67h Function DE0Ah

*VCPI*

#### *GET 8259 INTERRUPT VECTOR MAPPINGS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE0Ah | AH = 00h successful<br>BX = first vector used by master 8259 (IRQ0)<br>CX = first vector used by slave 8259 (IRQ8)<br>AH nonzero: failed |

**Details:** CX is undefined in systems without a slave 8259.

**See also:** Function DE0Bh, Phar Lap INT 21h Function 250Ch (Chapter 33), DPMI INT 31h Function 0400h (Chapter 30)

---

### INT 67h Function DE0Bh

*VCPI*

#### *SET 8259 INTERRUPT VECTOR MAPPINGS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE0Bh | AH = 00h successful |
| BX = first vector used by master 8259 | AH nonzero: failed |
| CX = first vector used by slave 8259 | |
| interrupts disabled | |

**Details:** This call merely informs the server that the client has changed the interrupt mappings. The client may not change the mappings if they have already been changed by the server or another client, and is responsible for restoring the original mappings before terminating.

**See also:** Function DE0Ah, Cloaking INT 2Ch Function 002Ah (Chapter 32)

---

### INT 67h Function DE0Ch

*VCPI*

#### *SWITCH TO PROTECTED MODE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE0Ch | interrupts disabled |
| ESI = linear address in first megabyte of values for system registers | GDTR, IDTR, LDTR, TR loaded |
| interrupts disabled | SS:ESP must have at least 16 bytes space, and the entry point is required to set up a new stack before enabling interrupts |
| | EAX, ESI, DS, ES, FS, GS destroyed |

**See also:** QEMM Function 3Fh (Table 39-27), BIOS INT 15h Function 89h (Chapter 4), PC-MOS/386 INT D4h Function 10h (Chapter 49)

**Details:** In protected mode, calling the protected-mode VCPI entry point with

DS = segment selector mapping entire linear address space obtained via Function DE01h

SS:ESP in first megabyte of linear memory

STACK:
QWORD return address from FAR call to 32-bit segment
DWORD EIP
DWORD CS
DWORD reserved for EFLAGS
DWORD ESP
DWORD SS
DWORD ES
DWORD DS
DWORD FS
DWORD GS

and interrupts disabled,
will switch to virtual86 mode with interrupts disabled, all segment registers loaded, and EAX destroyed.

---

**29-2 Format of system register values for switch to protected mode:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | value for CR3 |
| 04h | DWORD | linear address in first megabyte of value for GDTR |
| 08h | DWORD | linear address in first megabyte of value for IDTR |
| 0Ch | WORD | value for LDTR |
| 0Eh | WORD | value for TR |
| 10h | PWORD | CS:EIP of protected mode entry-point |

## INT 67h Function DE0Ch

## CHAPTER 30

# *DOS Protected-Mode Interface (DPMI)*

DPMI is the intended successor to VCPI (Virtual Control Program Interface, see Chapter 29). It allows DOS programs to use the protected-mode features of the 80286 and later processors without compromising built-in system protections. Version 0.9 of the specification is mainly a description of how the DPMI host in Windows 3.x functions; version 1.0 adds features such as error codes to improve its operation. At the time of writing, only one vendor (Qualitas) supplied a DPMI 1.0-compliant host due to a chicken-and-egg problem: no one will write to the 1.0 specification until there are hosts, and no incentive to write 1.0-compliant hosts exists until programs actually use the additional features.

Although not itself a DOS extender, DPMI provides all the primitives necessary to implement one. As a result, numerous cross-references from this chapter to Chapters 33 through 37 appear.

---

### INT 2Fh Function 1686h *DPMI*
### *DETECT MODE*

**Registers at call:**
AX = 1686h

**Return Registers:**
AX = status
  0000h if operating in protected mode under DPMI (INT 31h available)
  nonzero if in real/V86 mode or no DPMI (INT 31h not available)

**See also:** Function 1687h

---

### INT 2Fh Function 1687h *DPMI*
### *INSTALLATION CHECK*

**Registers at call:**
AX = 1687h

**Return Registers:**
AX = 0000h if installed
  BX = flags
    bit 0: 32-bit programs supported
  CL = processor type (02h=80286, 03h=80386, 04h=80486)
  DH = DPMI major version
  DL = two-digit DPMI minor version (binary)
  SI = paragraphs of DOS extender private data
  ES:DI -> DPMI mode-switch entry point (see Table 30-1)
AX nonzero if not installed

**BUG:** The Windows NT March93 beta reports version 0090h (0.144).

**See also:** Function 1686h, INT 31h Function 0400h, DPMS Function 43E0h (Chapter 31), QDPMI Function DE01h (Chapter 63), DPMILOAD Function FB42h Subfunction 0001h (Chapter 36), Netroom INT 31h Function 5702h (Chapter 41), PC-MOS/386 INT D4h Function 10h (Chapter 49)

#### 30-1 Mode switch entry point calling convention:

**Registers at call:**

AX = flags
  bit 0: set if 32-bit program
ES = real mode segment of buffer for DPMI private data (ignored if SI was zero)

**Return Registers:**

CF set on error
  program still in real mode
  AX = error code (DPMI 1.0+)
    8011h unable to allocate all necessary descriptors
    8021h 32-bit program specified, but 16-bit DPMI host
CF clear if successful
  CS = 16-bit selector corresponding to real-mode CS
  SS = selector corresponding to real-mode SS (64K limit)
  DS = selector corresponding to real-mode DS (64K limit)
  ES = selector to program's PSP (256-byte limit)
  FS = GS = 0
  high word of ESP = 0 if 32-bit program
  program now in protected mode

**Details:** This entry point is only called for the initial switch to protected mode.

---

### INT 2Fh Function 168Ah

### *GET VENDOR-SPECIFIC API ENTRY POINT*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**

AX = 168Ah
DS:(E)SI = selector:offset of ASCI vendor name

**Return Registers:**

AL = status
  00h successful
    ES:(E)DI -> extended API entry point
  8Ah unsuccessful

**Details:** The vendor name is used to determine which entry point to return; it is case-sensitive.

32-bit applications use ESI and EDI, 16-bit applications use SI and DI.

This call is present but not documented for DPMI 0.9.

The Borland C++ 3.1 DPMILOAD does not handle requests for entry points other than the MS-DOS one gracefully, producing an unhandled exception report.

**See also:** INT 31h Function 0A00h, Netroom INT 31h Function 57h (Chapter 41)

#### 30-2 Vendor names:

| | |
|---|---|
| "MS-DOS" | *unknown* and 386MAX v6.00+ |
| "386MAX" | 386MAX v6.00+ |
| "HELIX_DPMI" | Helix Netroom's DPMI server |

---

### INT 31h Function 0000h

### *ALLOCATE LDT DESCRIPTORS*

*DPMI 0.9+*
*Protected Mode*

**Purpose:** Allocate a specified number of contiguous descriptors in the calling task's Local Descriptor Table.

**Registers at call:**

AX = 0000h
CX = number of descriptors to allocate

**Return Registers:**

CF clear if successful
  AX = base selector
CF set on error
  AX = error code (DPMI 1.0+) (see Table 30-3)

**Details:** The base and limit of the returned descriptors will be 0, and the type will be "data".

Add the value returned by Function 0003h to move to subsequent descriptors if multiple descriptors were allocated.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Functions 0001h and 000Dh, FlashTek INT 21h Function 3501h (Chapter 33)

**30-3 Values for error code:**

| | |
|---|---|
| 0000h-7FFFh | DOS error passed through by DPMI |
| 8001h | unsupported function |
| 8002h | object in wrong state for function |
| 8003h | system integrity would be endangered |
| 8004h | deadlock detected |
| 8005h | pending serialization request cancelled |
| 8010h | out of DPMI internal resources |
| 8011h | descriptor unavailable |
| 8012h | linear memory unavailable |
| 8013h | physical memory unavailable |
| 8014h | backing store unavailable |
| 8015h | callback unavailable |
| 8016h | handle unavailable |
| 8017h | maximum lock count exceeded |
| 8018h | shared memory already serialized exclusively by another client |
| 8019h | shared memory already serialized shared by another client |
| 8021h | invalid value for numeric or flag parameter |
| 8022h | invalid segment selector |
| 8023h | invalid handle |
| 8024h | invalid callback |
| 8025h | invalid linear address |
| 8026h | request not supported by hardware |

---

### INT 31h Function 0001h
### *FREE LDT DESCRIPTOR*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0001h
BX = selector to free

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (DPMI 1.0+) (8022h) (see Table 30-3)

**Details:** Only one descriptor is freed per call.
The program's initial CS, DS, and SS descriptors may be freed.
(DPMI 1.0+) Any segment registers containing the freed selector are set to 0000h.
This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Functions 0000h, 000Ah, and 000Dh; FlashTek INT 21h Function 3502h (Chapter 33)

---

### INT 31h Function 0002h
### *SEGMENT TO DESCRIPTOR*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0002h
BX = real mode segment

**Return Registers:**
CF clear if successful
AX = selector corresponding to real mode segment (64K limit)
CF set on error
AX = error code (DPMI 1.0+) (8011h) (see Table 30-3)

**Details:** Multiple calls for the same real mode segment return the same selector.
The returned descriptor can never be modified or freed.
This function is not supported by MS Windows 3.0 in Standard mode.

## INT 31h Function 0003h
## *GET NEXT SELECTOR INCREMENT VALUE*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0003h

**Return Registers:**
CF clear
AX = value to add to get next sequential selector

**Details:** The increment will be a power of two.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Function 0000h

## INT 31h Function 0004h
## *LOCK SELECTOR*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0004h
BX = selector to lock (prevent paging)

**Return Registers:**
*unknown*

**Details:** Although marked as reserved in versions 0.9 and 1.0 of the DPMI specification, this function is called by MS Windows TASKMAN, PROGMAN, and KERNEL.

**See also:** Functions 0005h and 0600h

## INT 31h Function 0005h
## *UNLOCK SELECTOR*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0005h
BX = selector to unlock (permit paging)

**Return Registers:**
*unknown*

**Details:** Although marked as reserved in versions 0.9 and 1.0 of the DPMI specification, this function is called by MS Windows TASKMAN, PROGMAN, and KERNEL.

**See also:** Functions 0004h and 0601h

## INT 31h Function 0006h
## *GET SEGMENT BASE ADDRESS*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0006h
BX = selector

**Return Registers:**
CF clear if successful
CX:DX = linear base address of segment
CF set on error
AX = error code (DPMI 1.0+) (8022h) (see Table 30-3)

**Details:** This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Function 0007h, FlashTek INT 21h Function 3504h (Chapter 33)

## INT 31h Function 0007h
## *SET SEGMENT BASE ADDRESS*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0007h
BX = selector
CX:DX = linear base address

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (DPMI 1.0+) (8022h,8025h) (see Table 30-3)

**Details:** Only modify descriptors allocated with INT 31h Function 0000h.

Only the low 24 bits of the address will be used by 16-bit DPMI implementations even on a 386 or higher.

DPMI 1.0+ automatically reloads any segment registers containing the selector being modified.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Functions 0006h, 0008h, 0009h, and 000Ch; FlashTek INT 21h Function 3503h (Chapter 33); OS/286 INT 21h Function E9h (Chapter 34), Cloaking INT 2Ch Function 0002h (Chapter 32)

---

### INT 31h Function 0008h

***SET SEGMENT LIMIT***

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0008h
BX = selector
CX:DX = segment limit

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (DPMI 1.0+) (8021h,8022h,8025h) (see Table 30-3)

**Details:** CX must be zero for 16-bit DPMI implementations.

Limits greater than 1MB must be page aligned (low 12 bits set).

Only modify descriptors allocated with INT 31h Function 0000h.

DPMI 1.0+ automatically reloads any segment registers containing the selector being modified.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Functions 0007h, 0009h, and 000Ch; FlashTek INT 21h Function 3505h (Chapter 33); OS/286 INT 21h Function E9h (Chapter 34), Cloaking INT 2Ch Function 0003h (Chapter 32)

---

### INT 31h Function 0009h

***SET DESCRIPTOR ACCESS RIGHTS***

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0009h
BX = selector
CL = access rights/type byte
CH = 80386 extended rights/type byte (32-bit DPMI implementations only)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (DPMI 1.0+) (8021h,8022h,8025h) (see Table 30-3)

**Details:** If the Present bit is clear, CL bits 0-3 may have any value.

DPMI 1.0+ automatically reloads any segment registers containing the selector being modified.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Functions 0007h, 0008h, and 000Ch; Phar Lap INT 21h Function 2514h (Chapter 33); Cloaking INT 2Ch Functions 0004h and 0005h (Chapter 32)

---

### INT 31h Function 000Ah

***CREATE ALIAS DESCRIPTOR***

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 000Ah
BX = selector

**Return Registers:**
CF clear if successful
AX = new data selector
CF set on error
AX = error code (DPMI 1.0+) (8011h,8022h) (see Table 30-3)

**Details:** This call fails if the selector in BX is not a code segment or is invalid.

Use INT 31h Function 0001h to free new selector.

Future changes to the original selector will not be reflected in the returned alias selector.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Function 0001h

---

### INT 31h Function 000Bh

***GET DESCRIPTOR***

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 000Bh

BX = LDT selector
ES:(E)DI -> 8-byte buffer for copy of descriptor

**Return Registers:**
CF clear if successful
buffer filled
CF set on error
AX = error code (DPMI 1.0+) (8022h) (see Table 30-3)

**Details:** 16-bit programs use ES:DI as pointer, 32-bit must use ES:EDI.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Functions 000Ch and 000Eh

---

## INT 31h Function 000Ch
## *SET DESCRIPTOR*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 000Ch
BX = LDT selector
ES:(E)DI -> 8-byte buffer containing descriptor

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (DPMI 1.0+) (8021h,8022h,8025h) (see Table 30-3)

**Details:** 16-bit programs use ES:DI as pointer, 32-bit must use ES:EDI.

Only modify descriptors allocated with INT 31h Function 0000h.

DPMI 1.0+ automatically reloads any segment registers containing the selector being modified.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Function 000Bh

---

## INT 31h Function 000Dh
## *ALLOCATE SPECIFIC LDT DESCRIPTOR*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 000Dh
BX = LDT selector

**Return Registers:**
CF clear if successful
descriptor allocated
CF set on error
AX = error code (DPMI 1.0+) (8011h,8022h) (see Table 30-3)

**Details:** Free the returned descriptor with INT 31h Function 0001h.

The first 16 descriptors (04h-7Ch) are reserved for this function, but some may already be in use by other applications under DPMI 0.9; DPMI 1.0 guarantees 16 descriptors per client.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Functions 0000h and 0001h

---

## INT 31h Function 000Eh
## *GET MULTIPLE DESCRIPTORS*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 000Eh
CX = number of descriptors to copy
ES:(E)DI -> descriptor buffer (see Table 30-4)

**Return Registers:**
CF clear if successful
descriptors copied
CF set on error
AX = error code (8022h) (see Table 30-3)
CX = number of descriptors successfully copied

**Details:** 16-bit programs use ES:DI as pointer, 32-bit must use ES:EDI.

If the function fails, the first CX descriptors are valid; the remainder are not modified.

**See also:** Functions 000Bh and 000Fh

**30-4 Format of descriptor buffer entry (one per descriptor to get):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | selector (set by client) |
| 02h | QWORD | descriptor (set by host) |

### INT 31h Function 000Fh
### *SET MULTIPLE DESCRIPTORS*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 000Fh
CX = number of descriptors to copy
ES:(E)DI -> descriptor buffer (see Table 30-5)

**Return Registers:**
CF clear if successful
descriptors copied
CF set on error
AX = error code (8021h,8022h,8025h) (see Table 30-3)
CX = number of descriptors successfully copied

**Details:** 16-bit programs use ES:DI as pointer, 32-bit must use ES:EDI.

If the function fails, the first CX descriptors are valid; the remainder are not modified.

DPMI 1.0+ automatically reloads any segment registers containing a selector being modified.

**See also:** Functions 000Ch and 000Eh

**30-5 Format of descriptor buffer entry (one per descriptor to set):**

| Offset | Size | Description |
|---|---|---|
| 00h | WORD | selector |
| 02h | QWORD | descriptor |

### INT 31h Function 0100h
### *ALLOCATE DOS MEMORY BLOCK*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0100h
BX = number of paragraphs to allocate

**Return Registers:**
CF clear if successful
AX = real mode segment of allocated block
DX = first selector for allocated block
CF set on error
AX = DOS error code (07h,08h) (see Table 24-2)
(DPMI 1.0+) DPMI error code (8011h) (see Table 30-3)
BX = size (in paragraphs) of largest available block

**Details:** Multiple contiguous selectors are allocated for blocks of more than 64K if the caller is a 16-bit program.

Never modify or deallocate the returned descriptors.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Functions 0101h and 0501h

### INT 31h Function 0101h
### *FREE DOS MEMORY BLOCK*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0101h
DX = selector of block

**Return Registers:**
CF set if successful
CF set on error
AX = DOS error code (07h,09h) (see Table 24-2)

**Details:** All descriptors allocated for the block are automatically freed.

DPMI 1.0+ automatically zeros any segment registers containing a selector freed by this function.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Functions 0100h, 0102h, and 0502h

### INT 31h Function 0102h
### *RESIZE DOS MEMORY BLOCK*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0102h
BX = new block size in paragraphs
DX = selector of block

**Return Registers:**
CF clear if successful
CF set on error

AX = DOS error code (07h,08h,09h) (see Table 24-2)
(DPMI 1.0+) DPMI error code (8011h,8022h) (see Table 30-3)
BX = maximum block size (in paragraphs) possible

**Details:** Increasing the size of a block past a 64K boundary will fail if the next descriptor in the LDT is already in use.

Shrinking a block past a 64K boundary will cause some selectors to be freed; DPMI 1.0+ automatically zeros any segment registers containing a selector freed by this function.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Function 0100h

---

### INT 31h Function 0200h
### *GET REAL MODE INTERRUPT VECTOR*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0200h
BL = interrupt number

**Return Registers:**
CF clear
CX:DX = segment:offset of real mode interrupt handler

**Details:** The DPMI implementation is required to support all 256 vectors.

**See also:** Functions 0201h and 0204h, Phar Lap INT 21h Function 2503h (Chapter 33)

---

### INT 31h Function 0201h
### *SET REAL MODE INTERRUPT VECTOR*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0201h
BL = interrupt number
CX:DX = segment:offset of real mode handler

**Return Registers:**
CF clear

**Details:** All memory that may be touched by a hardware interrupt handler must be locked down with INT 31h Function 0600h.

**See also:** Functions 0200h, 0205h, and 0600h; Phar Lap INT 21h Function 2505h (Chapter 33)

---

### INT 31h Function 0202h
### *GET PROCESSOR EXCEPTION HANDLER VECTOR*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0202h
BL = exception number (00h-1Fh)

**Return Registers:**
CF clear if successful
CX:(E)DX = selector:offset of handler
CF set on error
AX = error code (DPMI 1.0+) (8021h) (see Table 30-3)

**Details:** 16-bit programs receive the pointer in CX:DX, 32-bit programs in CX:EDX.

DPMI 1.0+ supports this function only for backward compatibility; use Functions 0210h or 0211h instead.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Functions 0203h, 0210h, and 0211h; DPMILOAD INT 2Fh Function FB42h Subfunction 0021h (Chapter 36)

---

### INT 31h Function 0203h
### *SET PROCESSOR EXCEPTION HANDLER VECTOR*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0203h
BL = exception number (00h-1Fh)
CX:(E)DX = selector:offset of handler

**Return Registers:**
CF clear if successful
CF set on error

AX = error code (DPMI 1.0+) (8021h,8022h)
(see Table 30-3)

**Details:** 32-bit programs must supply an offset in EDX and use a 32-bit interrupt stack frame on chaining to the next exception handler.

The handler should return using a FAR return.

All fault stack frames contain an error code, but it is only valid for exceptions 08h and 0Ah-0Eh.

The handler will only be called if the exception occurs in protected mode and the DPMI host does not transparently handle the exception.

The handler may change certain values on the stack frame (see Tables 30-6 and 30-7).

DPMI 1.0+ supports this function only for backward compatibility; use Functions 0212h or 0213h instead.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Functions 0202h, 0212h, and 0213h; DPMILOAD INT 2Fh Function FB42h Subfunction 0022h (Chapter 36)

**30-6 Format of stack frame for 16-bit programs: (offset from SS:SP)**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | return CS:IP (do not change) |
| 04h | WORD | error code |
| 06h | DWORD | CS:IP of exception |
| 0Ah | WORD | flags |
| 0Ch | DWORD | SS:SP |

**30-7 Format of stack frame for 32-bit programs: (offset from SS:ESP)**

| Offset | Size | Description |
|---|---|---|
| 00h | DWORD | return EIP (do not change) |
| 04h | WORD | return CS selector (do not change) |
| 06h | WORD | reserved (do not change) |
| 08h | DWORD | error code |
| 0Ch | DWORD | EIP of exception |
| 10h | WORD | CS selector of exception |
| 12h | WORD | reserved (do not change) |
| 14h | DWORD | EFLAGS |
| 18h | DWORD | ESP |
| 1Ch | WORD | SS |
| 1Eh | WORD | reserved (do not change) |

---

### INT 31h Function 0204h

*DPMI 0.9+*

### *GET PROTECTED MODE INTERRUPT VECTOR*

*Protected Mode*

| Registers at call: | Return Registers: |
|---|---|
| AX = 0204h | CF clear |
| BL = interrupt number | CX:(E)DX = selector:offset of handler |

**Details:** 16-bit programs use CX:DX, 32-bit programs use CX:EDX.

DPMI implementations are required to support all 256 vectors.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Functions 0200h and 0205h, Phar Lap INT 21h Function 2502h (Chapter 33), Cloaking INT 2Ch Function 0006h (Chapter 32), DPMILOAD INT 2Fh Function FB42h Subfunction 0024h (Chapter 36)

---

### INT 31h Function 0205h
### *SET PROTECTED MODE INTERRUPT VECTOR*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0205h
BL = interrupt number
CX:(E)DX = selector:offset of handler

**Return Registers:**
CF clear if successful
CF set on error
  AX = error code (DPMI 1.0+) (8022h) (see Table 30-3)

**Details:** 16-bit programs use CX:DX, 32-bit programs use CX:EDX.

32-bit programs must use a 32-bit interrupt stack frame when chaining to the next handler.

DPMI implementations are required to support all 256 vectors.

Hardware interrupts are reflected to the virtual machine's primary client, software interrupts to the current client.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Functions 0201h and 0204h, Phar Lap INT 21h Function 2504h (Chapter 33), Cloaking INT 2Ch Function 0007h (Chapter 32), DPMILOAD INT 2Fh Function FB42h Subfunction 0025h (Chapter 36)

---

### INT 31h Function 0210h
### *GET PROTECTED MODE EXTENDED PROCESSOR EXCEPTION HANDLER*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 0210h
BL = exception number (00h-1Fh)

**Return Registers:**
CF clear if successful
  CX:(E)DX = selector:offset of exception handler
CF set on error
  AX = error code (8021h) (see Table 30-3)

**Details:** The DPMI host reflects the exception to the current client's handler.

The DPMI host returns CX:DX for 16-bit programs and CX:EDX for 32-bit programs.

**See also:** Functions 0202h, 0211h, and 0212h

---

### INT 31h Function 0211h
### *GET REAL MODE EXTENDED PROCESSOR EXCEPTION HANDLER*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 0211h
BL = exception number (00h-1Fh)

**Return Registers:**
CF clear if successful
  CX:(E)DX = selector:offset of exception handler
CF set on error
  AX = error code (8021h) (see Table 30-3)

**Details:** This function returns the address of the protected-mode handler for a real-mode exception.

The DPMI host performs a switch to protected mode, reflects the exception to the virtual machine's primary client, and returns to real mode on the handler's completion.

**See also:** Functions 0202h, 0210h, and 0213h

---

### INT 31h Function 0212h
### *SET PROTECTED MODE EXTENDED PROCESSOR EXCEPTION HANDLER*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 0212h
BL = exception or fault number (00h-1Fh)
CX:(E)DX = exception handler selector:offset

**Return Registers:**
CF clear if successful
CF set on error
  AX = error code (8021h,8022h) (see Table 30-3)

**Details:** The DPMI host sends the exception to the current client's handler.

**See also:** Functions 0203h, 0210h, and 0213h

---

**INT 31h Function 0213h** *DPMI 1.0+*

***SET REAL MODE EXTENDED PROCESSOR EXCEPTION HANDLER*** *Protected Mode*

**Purpose:** Specify the address of a protected-mode handler for a real-mode exception.

**Registers at call:**
AX = 0213h
BL = exception or fault number
  (00h-1Fh)
CX:(E)DX = exception handler
  selector:offset

**Return Registers:**
CF clear if successful
CF set on error
    AX = error code (8021h,8022h) (see Table 30-3)

**Details:** The DPMI host performs a switch to protected mode, reflects the exception to the virtual machine's primary client, and returns to real mode on the handler's completion.

**See also:** Functions 0203h, 0211h, and 0212h

---

**INT 31h Function 0300h** *DPMI 0.9+*

***SIMULATE REAL MODE INTERRUPT*** *Protected Mode*

**Registers at call:**
AX = 0300h
BL = interrupt number
BH = flags
    bit 0: reset the interrupt controller
      and A20 line (DPMI 0.9)
        reserved, must be 0 (DPMI 1.0+)
    others must be 0
CX = number of words to copy from
  protected mode to real mode stack
ES:(E)DI = selector:offset of real
  mode call structure (see Table 30-8)

**Return Registers:**
CF clear if successful
    real mode call structure modified (all fields except
SS:SP, CS:IP filled with return values from real
  mode interrupt)
CF set on error
    AX = error code (DPMI 1.0+)
      (8012h,8013h,8014h,8021h) (see Table 30-3)
protected mode stack unchanged

**Details:** 16-bit programs use ES:DI as pointer, 32-bit programs use ES:EDI.

The CS:IP in the real mode call structure is ignored for this call; instead, the indicated interrupt vector is used for the address.

The flags in the call structure are pushed on the real mode stack to form an interrupt stack frame, and the trace and interrupt flags are clear on entry to the handler.

DPMI will provide a small (30 words) real mode stack if SS:SP is zero.

The real mode handler must return with the stack in the same state as it was on being called.

**See also:** Function 0302h, Phar Lap INT 21h Function 2511h (Chapter 33), OS/286 INT 21h Function E3h (Chapter 34), Cloaking INT 2Ch Function 0026h (Chapter 32), DPMILOAD INT 2Fh Function FB42h Subfunction 000Dh (Chapter 36)

**30-8 Format of real mode call structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | EDI |
| 04h | DWORD | ESI |
| 08h | DWORD | EBP |
| 0Ch | DWORD | reserved (00h) |
| 10h | DWORD | EBX |
| 14h | DWORD | EDX |
| 18h | DWORD | ECX |
| 1Ch | DWORD | EAX |
| 20h | WORD | flags |
| 22h | WORD | ES |
| 24h | WORD | DS |
| 26h | WORD | FS |
| 28h | WORD | GS |
| 2Ah | WORD | IP |
| 2Ch | WORD | CS |
| 2Eh | WORD | SP |
| 30h | WORD | SS |

### INT 31h Function 0301h
### *CALL REAL MODE PROCEDURE WITH FAR RETURN FRAME*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0301h
BH = flags
  bit 0: reset the interrupt controller and A20 line (DPMI 0.9)
    reserved, must be 0 (DPMI 1.0+)
  others must be 0
CX = number of words to copy from protected mode to real mode stack
ES:DI / ES:EDI = selector:offset of real mode call structure (see Table 30-8)

**Return Registers:**
CF clear if successful
  real mode call structure modified (all fields except SS:SP, CS:IP filled with return values from real mode interrupt)
CF set on error
  AX = error code (DPMI 1.0+) (8012h,8013h,8014h,8021h) (see Table 30-3)
protected mode stack unchanged

**Details:** 16-bit programs use ES:DI as pointer, 32-bit programs use ES:EDI.

The real-mode procedure must exit with a FAR return.

DPMI will provide a small (30 words) real mode stack if SS:SP is zero.

The real mode handler must return with the stack in the same state as it was on being called.

**See also:** Functions 0300h and 0302h, Phar Lap INT 21h Function 250Eh (Chapter 33), OS/286 INT 21h Function E1h (Chapter 34), Cloaking INT 2Ch Function 0025h (Chapter 32)

---

### INT 31h Function 0302h
### *CALL REAL MODE PROCEDURE WITH IRET FRAME*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0302h
BH = flags
  bit 0: reset the interrupt controller and A20 line (DPMI 0.9)
    reserved, must be 0 (DPMI 1.0+)
  others must be 0
CX = number of words to copy from protected mode to real mode stack
ES:DI / ES:EDI = selector:offset of real mode call structure (see Table 30-8)

**Return Registers:**
CF clear if successful
  real mode call structure modified (all fields except SS:SP, CS:IP filled with return values from real mode interrupt)
CF set on error
  AX = error code (DPMI 1.0+) (8012h,8013h,8014h,8021h) (see Table 30-3)
protected mode stack unchanged

**Details:** 16-bit programs use ES:DI as pointer, 32-bit programs use ES:EDI.

The flags in the call structure are pushed on the real mode stack to form an interrupt stack frame, and the trace and interrupt flags are clear on entry to the handler.

The real mode procedure must exit with an IRET.

DPMI will provide a small (30 words) real mode stack if SS:SP is zero.

The real mode handler must return with the stack in the same state as it was on being called.

**See also:** Function 0300h

---

### INT 31h Function 0303h
### *ALLOCATE REAL MODE CALLBACK ADDRESS*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0303h
DS:SI / DS:ESI = selector:offset of procedure to call
ES:DI / ES:EDI = selector:offset of real mode call structure (see Table 30-8)

**Return Registers:**
CF clear if successful
  CX:DX = segment:offset of real mode call address
CF set on error
  AX = error code (DPMI 1.0+) (8015h) (see Table 30-3)

**Details:** The real mode call structure is static, causing reentrancy problems; its contents are valid only at the time of a callback.

The called procedure must modify the real mode CS:IP before returning.

Values are returned to real mode by modifying the real mode call structure.

DPMI hosts must provide at least 16 callbacks per client.

**See also:** Functions 0304h and 0C00h

### 30-9 Callback procedure calling convention:

**Registers at call:**
DS:(E)SI = selector:offset of real mode SS:SP
ES:(E)DI = selector:offset of real mode call structure
SS:(E)SP = locked protected mode API stack
interrupts disabled

**Return Registers:**
(with IRET)
ES:(E)DI = selector:offset of real mode call structure to restore

---

## INT 31h Function 0304h

## *FREE REAL MODE CALLBACK ADDRESS*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0304h
CX:DX = real mode callback address

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (DPMI 1.0+) (8024h) (see Table 30-3)

**See also:** Function 0303h

---

## INT 31h Function 0305h

## *GET STATE SAVE/RESTORE ADDRESSES*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0305h

**Return Registers:**
CF clear
AX = size in bytes of state buffer
BX:CX = real mode address of procedure to save/restore state
SI:(E)DI = protected mode procedure to save/restore state

**Details:** The buffer size will be zero if it is not necessary to preserve state.

16-bit programs should call SI:DI, 32-bit programs should call SI:EDI.

This function is only needed if using the raw mode switch service.

**See also:** Function 0306h

### 30-10 State-save procedure calling convention:

**Registers at call:**
AL = direction
  00h save state
  01h restore state
ES:(E)DI -> state buffer

**Return Registers:**
all registers preserved

---

## INT 31h Function 0306h

## *GET RAW MODE SWITCH ADDRESSES*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0306h

**Return Registers:**
CF clear
BX:CX -> procedure to switch from real to protected mode
SI:(E)DI -> procedure to switch from protected to real mode

**Details:** 16-bit programs should jump to SI:DI, 32-bit programs should use SI:EDI.

The caller must save and restore the state of the task with Function 0305h.

This function is not supported by MS Windows 3.0 in Standard mode.

**See also:** Function 0305h

---

**30-11 Values with which to JUMP at mode-switch procedures:**

AX = new DS
CX = new ES
DX = new SS
(E)BX = new (E)SP
SI = new CS
(E)DI = new (E)IP

**Notes:** (E)BP is preserved across the call, but (E)AX, (E)BX, (E)CX, (E)DX, (E)SI, and (E)DI will be undefined; FS and GS will be 0000h.

Interrupts will stay disabled during the entire mode switch if they are disabled on entry to the mode-switch procedure.

---

## INT 31h Function 0400h
## *GET DPMI VERSION*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0400h

**Return Registers:**
CF clear
AH = major version of DPMI spec supported
AL = two-digit minor version of DPMI spec supported
BX = flags
- bit 0: running under an 80386 (32-bit) implementation
- bit 1: processor returns to real mode for reflected interrupts instead of V86 mode
- bit 2: virtual memory supported
- bit 3: reserved (undefined)
- others reserved (zero)

CL = processor type (02h=80286, 03h=80386, 04h=80486)
DH = current value of virtual master interrupt controller base interrupt
DL = current value of virtual slave interrupt controller base interrupt

**See also:** Function 0401h, INT 2Fh Function 1687h, Phar Lap INT 21h Function 250Ch (Chapter 33), VCPI INT 67h Function DE0Ah (Chapter 29)

---

## INT 31h Function 0401h
## *GET DPMI CAPABILITIES*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 0401h
ES:(E)DI -> 128-byte buffer for host description (see Table 30-13)

**Return Registers:**
CF clear if successful
- AX = capabilities (see Table 30-12)
- CX = reserved (00h)
- DX = reserved (00h)
- buffer filled

CF set on error (DPMI 0.9 only)

**See also:** Function 0400h

### 30-12 Bitfields for capabilities:

| | |
|---|---|
| bit 0 | paged accessed/dirty supported (see Functions 0506h and 0507h) |
| bit 1 | exceptions restartability supported |
| bit 2 | device mapping supported (see Function 0508h) |
| bit 3 | conventional memory mapping supported (see Function 0509h) |
| bit 4 | demand zero-fill supported |
| bit 5 | write-protect client capability supported |
| bit 6 | write-protect host capability supported |
| bits 7-15 | reserved |

### 30-13 Format of host description:

| Offset | Size | Description |
|---|---|---|
| 00h | BYTE | host major version number |
| 01h | BYTE | host minor version number |
| 02h | 126 BYTEs | ASCIZ host vendor name |

---

**INT 31h Function 0500h** *DPMI 0.9+*

***GET FREE MEMORY INFORMATION*** *Protected Mode*

**Registers at call:**
AX = 0500h
ES:(E)DI -> buffer for memory information (see Table 30-14)

**Return Registers:**
CF clear

**Details:** 16-bit programs use ES:DI, 32-bit programs use ES:EDI.

This function must be considered advisory because other applications may affect the results at any time after the call.

Fields not supported by the DPMI implementation are filled with FFFFFFFFh.

DPMI 1.0+ supports this function solely for backward compatibility; use Function 050Bh instead.

**See also:** Functions 0501h and 0604h

### 30-14 Format of memory information:

| Offset | Size | Description |
|---|---|---|
| 00h | DWORD | largest available block in bytes |
| 04h | DWORD | maximum unlocked page allocation |
| 08h | DWORD | maximum locked page allocation |
| 0Ch | DWORD | total linear address space in pages |
| 10h | DWORD | total unlocked pages |
| 14h | DWORD | free pages |
| 18h | DWORD | total physical pages |
| 1Ch | DWORD | free linear address space in pages |
| 20h | DWORD | size of paging file/partition in pages |
| 24h | 12 BYTEs | reserved |

---

**INT 31h Function 0501h** *DPMI 0.9+*

***ALLOCATE MEMORY BLOCK*** *Protected Mode*

**Registers at call:**
AX = 0501h
BX:CX = size in bytes

**Return Registers:**
CF clear if successful
BX:CX = linear address of block
SI:DI = memory block handle for resizing and freeing block
CF set on error

AX = error code (DPMI 1.0+)
(8012h-8014h,8016h,8021h) (see Table 30-3)

**Details:** No selectors are allocated.

The memory block is allocated unlocked (it can be locked with Function 0600h).

Allocations are often page granular (see Function 0604h).

Under MS Windows 3.10 Enhanced mode with paging enabled, it is possible for this function to fail even if Function 0500h indicates that enough memory is available.

**See also:** Functions 0000h, 0100h, 0500h, 0502h, 0503h, 0504h, 0D00h; DPMILOAD INT 2Fh Function FB42h Subfunction 0002h (Chapter 36)

---

### INT 31h Function 0502h

### *FREE MEMORY BLOCK*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0502h
SI:DI = handle of memory block

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (DPMI 1.0+) (8023h) (see Table 30-3)

**Details:** Any selectors allocated for the memory block must also be freed, preferably before freeing the memory block.

**See also:** Functions 0001h, 0101h, 0501h, and 0D01h

---

### INT 31h Function 0503h

### *RESIZE MEMORY BLOCK*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0503h
BX:CX = new size in bytes (nonzero)
SI:DI = handle of memory block

**Return Registers:**
CF clear if successful
BX:CX = new linear address
SI:DI = new handle of memory block
CF set on error
AX = error code (DPMI 1.0+)
(8012h-8014h,8016h,8021h,8023h) (see Table 30-3)

**Details:** Any selectors pointing at the block must be updated.

The previous memory block handle becomes invalid.

An error is returned if the new size is 0.

**See also:** Functions 0102h, 0501h, and 0505h

---

### INT 31h Function 0504h

### *ALLOCATE LINEAR MEMORY BLOCK*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 0504h
EBX = page-aligned linear address of memory block (00000000h if any address is acceptable)
ECX = size in bytes (nonzero)
EDX = flags
bit 0: set to create committed pages instead of uncommitted pages
bits 1-31 reserved (0)

**Return Registers:**
CF clear if successful
EBX = linear address of memory block
ESI = memory block handle
CF set on error
AX = error code
(8001h,8012h-8014h,8016h,8021h,8025h)
(see Table 30-3)

**Details:** This function is only supported by 32-bit DPMI hosts, but may be used by 16-bit clients.

**See also:** Functions 0501h and 0505h

---

### INT 31h Function 0505h

### *RESIZE LINEAR MEMORY BLOCK*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 0505h
ESI = memory block handle
ECX = new size in bytes (nonzero)
EDX = flags
bit 0: create committed pages
bit 1: segment descriptor update required
ES:EBX -> buffer containing array of WORDs with selectors
EDI = number of selectors in array
bits 2-31 reserved (0)

**Return Registers:**
CF clear if successful
EBX = new linear base address
ESI = new memory block handle
CF set on error
AX = error code (8001h,8012h-8014h,8016h,8021h,8023h) (see Table 30-3)

**Details:** This function is only supported by 32-bit DPMI hosts, but may be used by 16-bit clients.

The old memory block handle becomes invalid.

If EDX bit 1 is set and the block's base address is changed, DPMI updates all descriptors for selectors in the update buffer which fall within the memory block.

**See also:** Functions 0503h and 0504h

---

### INT 31h Function 0506h

### *GET PAGE ATTRIBUTES*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 0506h
ESI = memory block handle
EBX = offset in memory block of first page
ECX = number of pages
ES:EDX -> array of WORDs to hold page attributes (see Table 30-15)

**Return Registers:**
CF clear if successful
buffer filled
CF set on error
AX = error code (8001h,8023h,8025h) (see Table 30-3)

**Details:** This function is only supported by 32-bit DPMI hosts, but may be used by 16-bit clients.

If EBX is not page-aligned, it will be rounded down.

**See also:** Functions 0504h and 0507h, Phar Lap INT 21h Function 251Dh (Chapter 33), OS/386 INT 21h Function EB00h (Chapter 34)

**30-15 Format of page attribute words:**

| | |
|---|---|
| bits 0-2 | page type<br>000 uncommitted<br>001 committed<br>010 mapped (sees Functions 0508h and 0509h)<br>other currently unused |
| bit 3 | page is read/write rather than read-only |
| bit 4 | accessed/dirty bits supplied in bits 5 and 6 |
| bit 5 | page has been accessed (only valid if bit 4 set) |
| bit 6 | page has been written (only valid if bit 4 set) |
| bits 7-15 | reserved (0) |

---

### INT 31h Function 0507h

### *MODIFY PAGE ATTRIBUTES*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 0507h
ESI = memory block handle

EBX = offset in memory block of first page
ECX = number of pages
ES:EDX -> array of WORDs with new page attributes (see Table 30-15)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (8001h,8002h,8013h,8014h,8021h,8023h, 8025h) (see Table 30-3)
ECX = number of pages which have been set

**Details:** This function is only supported by 32-bit DPMI hosts, but may be used by 16-bit clients.
If EBX is not page-aligned, it will be rounded down.

**See also:** Functions 0504h and 0506h, Phar Lap INT 21h Function 251Eh (Chapter 33)

---

### INT 31h Function 0508h
### *MAP DEVICE IN MEMORY BLOCK*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 0508h
ESI = memory block handle
EBX = page-aligned offset within memory block of page(s) to be mapped
ECX = number of pages to map
EDX = page-aligned physical address of device

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (8001h,8003h,8023h,8025h) (see Table 30-3)

**Details:** This function is only supported by 32-bit DPMI hosts, but may be used by 16-bit clients.
Support of this function is optional; hosts are also allowed to support the function for some devices but not others.

**See also:** Functions 0504h, 0509h, 0800h, and 0801h

---

### INT 31h Function 0509h
### *MAP CONVENTIONAL MEMORY IN MEMORY BLOCK*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 0509h
ESI = memory block handle
EBX = page-aligned offset within memory block of page(s) to map
ECX = number of pages to map
EDX = page-aligned linear address of conventional (below 1M) memory

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (8001h,8003h,8023h,8025h) (see Table 30-3)

**Details:** This function is only supported by 32-bit DPMI hosts, but may be used by 16-bit clients.
Support of this function is optional.

**See also:** Functions 0504h, 0508h, and 0801h

---

### INT 31h Function 050Ah
### *GET MEMORY BLOCK SIZE AND BASE*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 050Ah
SI:DI = memory block handle

**Return Registers:**
CF clear if successful
SI:DI = size in bytes
BX:CX = base address
CF set on error
AX = error code (8023h) (see Table 30-3)

**See also:** Functions 0501h and 0504h

---

## INT 31h Function 050Bh

### *GET MEMORY INFORMATION*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 050Bh
ES:(E)DI -> 128-byte buffer for memory information (see Table 30-16)

**Return Registers:**
CF clear if successful
CF set on error (DPMI 0.9 only)

**Details:** 16-bit programs use ES:DI, 32-bit programs must use ES:EDI.
**See also:** Function 0500h

**30-16 Format of memory information:**
DPMI;memory information
DPMI;virtual memory
DPMI;physical memory

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | total allocated bytes of physical memory controlled by host |
| 04h | DWORD | total allocated bytes of virtual memory controlled by host |
| 08h | DWORD | total available bytes of virtual memory controlled by host |
| 0Ch | DWORD | total allocated bytes of virtual memory for curr virtual mach |
| 10h | DWORD | total available bytes of virtual memory for curr virtual mach |
| 14h | DWORD | total allocated bytes of virtual memory for current client |
| 18h | DWORD | total available bytes of virtual memory for current client |
| 1Ch | DWORD | total locked bytes for current client |
| 20h | DWORD | maximum locked bytes for current client |
| 24h | DWORD | highest linear address available to current client |
| 28h | DWORD | largest available memory block in bytes |
| 2Ch | DWORD | minimum allocation unit in bytes |
| 30h | DWORD | allocation alignment unit size in bytes |
| 34h | 76 BYTEs | reserved (00h) |

---

## INT 31h Function 0600h

### *LOCK LINEAR REGION*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0600h
BX:CX = starting linear address
SI:DI = size of region in bytes

**Return Registers:**
CF clear if successful
CF set on error
none of the memory is locked
AX = error code (DPMI 1.0+) (8013h,8017h,8025h) (see Table 30-3)

**Details:** Pages at the beginning and end will be locked if the region overlaps them.

This function may be called multiple times for a given page; the DPMI host keeps a lock count for each page.

**See also:** Functions 0004h and 0601h, Phar Lap INT 21h Function 251Ah (Chapter 33), OS/386 INT 21h Function EB06h (Chapter 34)

---

## INT 31h Function 0601h

### *UNLOCK LINEAR REGION*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0601h
BX:CX = starting linear address
SI:DI = size of region in bytes

**Return Registers:**
CF clear if successful
CF set on error
none of the memory is unlocked
AX = error code (DPMI 1.0+) (8002h,8025h) (see Table 30-3)

**Details:** Pages at the beginning and end will be unlocked if the region overlaps them.

Memory whose lock count has not reached zero remains locked.

**See also:** Functions 0005h and 0600h, Phar Lap INT 21h Function 251Bh (Chapter 33), OS/386 INT 21h Function EB07h (Chapter 34)

---

### INT 31h Function 0602h
### *MARK REAL MODE REGION AS PAGEABLE*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0602h
BX:CX = starting linear address
SI:DI = size of region in bytes

**Return Registers:**
CF clear if successful
CF set on error
 none of the memory is made pageable
 AX = error code (DPMI 1.0+) (8002h,8025h) (see Table 30-3)

**Details:** The application must relock all unlocked real mode memory before terminating under DPMI 0.9; DPMI 1.0+ automatically relocks real mode memory.

Pages at the beginning and end will be unlocked if the region overlaps them.

Pageability of real mode pages is binary, not a count.

**See also:** Functions 0600h and 0603h

---

### INT 31h Function 0603h
### *RELOCK REAL MODE REGION*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0603h
BX:CX = starting linear address
SI:DI = size of region in bytes

**Return Registers:**
CF clear if successful
CF set on error
 none of the memory is relocked
 AX = error code (DPMI 1.0+) (8002h,8013h,8025h) (see Table 30-3)

**Details:** Pages at the beginning and end will be relocked if the region overlaps them.

Pageability of real mode pages is binary, not a count.

**See also:** Function 0602h

---

### INT 31h Function 0604h
### *GET PAGE SIZE*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0604h

**Return Registers:**
CF clear if successful
 BX:CX = page size in bytes
CF set on error
 AX = error code (DPMI 1.0+)
  8001h unsupported, 16-bit host

**BUG:** The Borland C++ 3.1 DPMILOAD does not handle this function properly, returning with CF clear but BX and CX unchanged.

---

### INT 31h Function 0700h
### *MARK PAGES AS PAGING CANDIDATES*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0700h
BX:CX = starting linear page number
SI:DI = number of pages to mark as paging candidates

**Return Registers:**
*unknown*

**Details:** Although marked as reserved in versions 0.9 and 1.0 of the DPMI specification, this function is called by MS Windows TASKMAN, PROGMAN, and KERNEL.

**See also:** Functions 0701h and 0702h

---

### INT 31h Function 0701h

### *DISCARD PAGES*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0701h
BX:CX = starting linear page number
SI:DI = number of pages to discard

**Return Registers:**
*unknown*

**Details:** Although marked as reserved in versions 0.9 and 1.0 of the DPMI specification, this function is called by MS Windows TASKMAN, PROGMAN, and KERNEL.

**See also:** Functions 0700h and 0703h

---

### INT 31h Function 0702h

### *MARK PAGE AS DEMAND PAGING CANDIDATE*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0702h
BX:CX = starting linear address
SI:DI = number of bytes to mark as paging candidates

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (DPMI 1.0+) (8025h) (see Table 30-3)

**Details:** This function is advisory, and does not force immediate paging.

Partial pages will not be discarded.

**See also:** Functions 0700h and 0703h

---

### INT 31h Function 0703h

### *DISCARD PAGE CONTENTS*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0703h
BX:CX = starting linear address
SI:DI = number of bytes to mark as discarded

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (DPMI 1.0+) (8025h) (see Table 30-3)

**Details:** This function is advisory, and may be ignored by DPMI implementations.

Partial pages will not be discarded.

**See also:** Functions 0701h and 0702h

---

### INT 31h Function 0800h

### *PHYSICAL ADDRESS MAPPING*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0800h
BX:CX = physical address (should be above 1 MB)
SI:DI = size in bytes

**Return Registers:**
CF clear if successful
BX:CX = linear address which maps the requested physical memory
CF set on error
AX = error code (DPMI 1.0+) (8003h,8021h) (see Table 30-3)

**Details:** Implementations may refuse this call because it can circumvent protections.

The caller must build an appropriate selector for the memory.

Do not use this function for memory mapped in the first megabyte.

**See also:** Functions 0002h, 0508h, 0509h, and 0801h; Phar Lap INT 21h Function 250Ah (Chapter 33); OS/386 INT 21h Function EB05h (Chapter 34)

---

### INT 31h Function 0801h

### *FREE PHYSICAL ADDRESS MAPPING*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 0801h

BX:CX = linear address returned by Function 0800h

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (8025h) (see Table 30-3)

**Details:** This function should be called when done accessing the device mapped with Function 0800h.

**See also:** Functions 0508h, 0509h, and 0800h; OS/386 INT 21h Function EB03h (Chapter 34)

---

### INT 31h Function 0900h
### *GET AND DISABLE VIRTUAL INTERRUPT STATE*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0900h

**Return Registers:**
CF clear
virtual interrupts disabled
AL = previous state
(00h disabled, 01h enabled)
AH preserved

**Details:** The previous state may be restored simply by executing another INT 31h.

A CLI instruction may be used if the previous state is unimportant, but should be assumed to be very slow due to trapping by the host.

**See also:** Functions 0901h and 0902h

---

### INT 31h Function 0901h
### *GET AND ENABLE VIRTUAL INTERRUPT STATE*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0901h

**Return Registers:**
CF clear
virtual interrupts enabled
AL = previous state
(00h disabled, 01h enabled)
AH preserved

**Details:** The previous state may be restored simply by executing another INT 31h.

A STI instruction may be used if the previous state is unimportant, but should be assumed to be very slow due to trapping by the host.

**See also:** Functions 0900h and 0902h

---

### INT 31h Function 0902h
### *GET VIRTUAL INTERRUPT STATE*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0902h

**Return Registers:**
CF clear
AL = current state
(00h disabled, 01h enabled)

**Details:** This function should be used rather than PUSHF because that instruction yields the physical interrupt state rather than the per-client virtualized interrupt flag.

**See also:** Functions 0900h and 0901h

---

### INT 31h Function 0A00h
### *GET VENDOR SPECIFIC API ENTRY POINT*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0A00h
DS:SI / DS:ESI -> case-sensitive ASCIZ vendor name or identifier

**Return Registers:**
CF clear if successful
ES:DI / ES:EDI -> FAR extended API entry point
DS, FS, GS, EAX, EBX, ECX, EDX, ESI, EBP destroyed

**Details:** Extended API parameters are vendor-specific.

DPMI 1.0+ supports this function solely for backward compatibility; use INT 2Fh Function 168Ah instead.

This function is not supported by MS Windows 3.10, BC++ 3.1 DPMILOAD, or QDPMI v1.0x; use INT 2Fh Function 168Ah instead. It is supported by 386MAX v7.01.

**See also:** INT 2Fh Function 168Ah

---

### INT 31h Function 0B00h
### *SET DEBUG WATCHPOINT*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0B00h
BX:CX = linear address
DL = size (1,2,4 bytes)
DH = type (00h execute, 01h write, 02h read/write)

**Return Registers:**
CF clear if successful
BX = watchpoint handle
CF set on error
AX = error code (DPMI 1.0+) (8016h,8021h,8025h) (see Table 30-3)

**See also:** Functions 0212h and 0601h

---

### INT 31h Function 0B01h
### *CLEAR DEBUG WATCHPOINT*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0B01h
BX = watchpoint handle

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (DPMI 1.0+) (8023h) (see Table 30-3)

**Details:** The watchpoint handle is freed.

**See also:** Function 0B00h

---

### INT 31h Function 0B02h
### *GET STATE OF DEBUG WATCHPOINT*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0B02h
BX = watchpoint handle

**Return Registers:**
CF clear if successful
AX = status flags
bit 0: watch point has been executed since Functions 0B00h or 0B03h
CF set on error
AX = error code (DPMI 1.0+) (8023h) (see Table 30-3)

**See also:** Functions 0B00h and 0B03h

---

### INT 31h Function 0B03h
### *RESET DEBUG WATCHPOINT*

*DPMI 0.9+*
*Protected Mode*

**Registers at call:**
AX = 0B03h
BX = watchpoint handle

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (DPMI 1.0+) (8023h) (see Table 30-3)

**See also:** Function 0B02h

## INT 31h Function 0C00h
## *INSTALL RESIDENT HANDLER INITIALIZATION CALLBACK*

*DPMI 1.0+*
*Protected Mode*

**Purpose:** Calling this function declares an intent to provide resident protected mode services after terminating with Function 0C01h.

**Registers at call:**
AX = 0C00h
ES:(E)DI -> resident service provider structure (see Table 30-17)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (8015h,8021h,8025h) (see Table 30-3)

**See also:** Functions 0303h and 0C01h

### 30-17 Format of resident service provider structure:

| Offset | Size | Description |
|---|---|---|
| 00h | QWORD | descriptor for 16-bit data segment |
| 08h | QWORD | descriptor for 16-bit code segment (zeros if not supported) |
| 10h | WORD | offset of 16-bit callback procedure |
| 12h | 2 BYTEs | reserved |
| 14h | QWORD | descriptor for 32-bit data segment |
| 1Ch | QWORD | descriptor for 32-bit code segment (zeros if not supported) |
| 24h | DWORD | offset of 32-bit callback procedure |

## INT 31h Function 0C01h
## *TERMINATE AND STAY RESIDENT*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 0C01h
BL = return code
DX = number of paragraphs of DOS memory to reserve (0 or >= 6)

**Return Registers:**
never returns

**Details:** This function should only be used if the program will only provide services to other DPMI programs.

Any protected mode memory remains allocated to the program unless explicitly freed before this call.

The application must first call Function 0C00h or the program will simply be terminated.

**See also:** Function 0C00h, MS-DOS INT 21h Function 31h (Chapter 24)

## INT 31h Function 0D00h
## *ALLOCATE SHARED MEMORY*

*DPMI 1.0+*
*Protected Mode*

**Registers at call:**
AX = 0D00h
ES:(E)DI -> shared memory allocation request structure (see Table 30-18)

**Return Registers:**
CF clear if successful
request structure updated
CF set on error
AX = error code (8012h,8013h,8014h,8016h,8021h) (see Table 30-3)

**Details:** First 16 bytes of memory block will be initialized to zeros on the first allocation.

**See also:** Functions 0501h, 0D01h, and 0D02h

### 30-18 Format of shared memory allocation request structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | requested length of shared memory block in bytes |
| 04h | DWORD | (return) allocated length of block |
| 08h | DWORD | (return) shared memory handle |
| 0Ch | DWORD | (return) linear address of memory block |

| | | |
|---|---|---|
| 10h | PWORD | selector:offset32 of ASCIZ name for memory block (name max 128 bytes) |
| 16h | 2 BYTEs | reserved |
| 18h | 4 BYTEs | reserved (00h) |

---

### INT 31h Function 0D01h

*DPMI 1.0+*

### *FREE SHARED MEMORY*

*Protected Mode*

**Registers at call:**
AX = 0D01h
SI:DI = shared memory block handle

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (8023h) (see Table 30-3)

**Details:** The handle becomes invalid after this call.

DPMI maintains separate global and virtual machine use counts for each shared memory block; when the global use count reaches zero, the block is finally destroyed.

**See also:** Functions 0502h and 0D00h

---

### INT 31h Function 0D02h

*DPMI 1.0+*

### *SERIALIZE SHARED MEMORY*

*Protected Mode*

**Registers at call:**
AX = 0D02h
SI:DI = shared memory block handle
DX = flags
  bit 0: return immediately rather than suspending if serialization unavailable
  bit 1: shared rather than exclusive serialization
  bits 2-15 reserved (0)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (8004h,8005h,8017h-8019h,8023h) (see Table 30-3)

**Details:** An exclusive serialization blocks any other serialization attempts for the same block by another virtual machine; a shared serialization blocks attempts at exclusive serialization by another virtual machine.

Hosts are not required to detect deadlock.

A client's interrupt handler can cancel a serialization call which caused it to block by calling Function 0D03h.

**See also:** Functions 0D00h and 0D03h

---

### INT 31h Function 0D03h

*DPMI 1.0+*

### *FREE SERIALIZATION ON SHARED MEMORY*

*Protected Mode*

**Registers at call:**
AX = 0D03h
SI:DI = shared memory block handle
DX = flags
  bit 0: release shared serialization rather than exclusive serialization
  bit 1: free pending serialization
  bits 2-15 reserved (0)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (8002h,8023h) (see Table 30-3)

**See also:** Functions 0D00h and 0D02h

---

### INT 31h Function 0E00h

*DPMI 1.0+*

### *GET COPROCESSOR STATUS*

*Protected Mode*

**Registers at call:**
AX = 0E00h

**Return Registers:**
CF clear
AX = coprocessor status (see Table 30-19)

**Details:** This function is supported by 386MAX v6.01, which otherwise only supports DPMI 0.9.

**See also:** Function 0E01h

### 30-19 Bitfields for coprocessor status:

| | |
|---|---|
| bit 0 | numeric coprocessor enabled for current client |
| bit 1 | client is emulating coprocessor |
| bit 2 | numeric coprocessor is present |
| bit 3 | host is emulating coprocessor instructions |
| bits 4-7 | coprocessor type<br>0000 none<br>0010 80287<br>0011 80387<br>0100 80486 with numeric coprocessor<br>other reserved |
| bits 8-15 | not used |

## INT 31h Function 0E01h

***SET EMULATION***

*DPMI 1.0+*
*Protected Mode*

| Registers at call: | Return Registers: |
|---|---|
| AX = 0E01h | CF clear if successful |
| BX = coprocessor flag bits (see Table 30-20) | CF set on error<br>AX = error code (8026h) (see Table 30-3) |

**Details:** This function is supported by 386MAX v6.01, which otherwise only supports DPMI 0.9.

**See also:** Function 0E00h

### 30-20 Bitfields for coprocessor flag bits:

| | |
|---|---|
| bit 0 | enable numeric coprocessor for current client |
| bit 1 | client will emulate coprocessor |
| bits 2-15 | not used |

## CHAPTER 31

# DOS Protected Mode Services (DPMS)

DPMS is Novell's specification for allowing TSRs to reside almost entirely in protected-mode memory. In this respect, it is similar to Helix Software's Cloaking specification described in Chapter 32. The DPMS specification was still undergoing beta test at the time of this writing.

Novell describes DPMS as a DOS version-independent and Microsoft Windows-compatible API that provides protected mode support for device drivers and TSRs. Novell's DOS 7 (not released at the time of writing) is expected to include several DPMS client components including disk cache and compression drivers, a peer-to-peer server, and CD-ROM extensions.

---

### INT 2Fh Function 43E0h *DPMS*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = 43E0h | AX = 0000h if installed<br>ES:BX -> registration structure (see Table 31-1) |

**See also:** DPMI INT 2Fh Function 1687h (Chapter 30), Cloaking INT 2Ch (Chapter 32)

**31-1 Format of registration structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | real-mode API entry point (see Tables 31-2 through 31-19) |
| 04h | DWORD | 16-bit protected-mode API entry point (see Tables 31-2 through 32-19) |
| 08h | 8 BYTEs | reserved (0) |
| 10h | 8 BYTEs | blank-padded server OEM name |
| 18h | WORD | flags<br>bit 0: fast processor reset available (286 only)<br>bits 1-15 reserved (undefined) |
| 1Ah | 2 BYTEs | DPMS version (major,minor) |
| 1Ch | BYTE | CPU type (02h = 286, 03h = 386 or higher) |

**31-2 DPMS "call protected-mode procedure" calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 0100h<br>CX = number of words of stack to copy<br>ES:DI -> callup/down register structure (see Table 31-21) | CF clear if successful<br>CF set on error<br>AX = error code (see Table 31-20) |

**31-3 DPMS "call real-mode procedure (RETF return)" calling convention:**

**Registers at call:**

AX = 0101h

CX = number of words of stack to copy

ES:DI -> callup/down register structure (see Table 31-21)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 31-20)

---

**31-4 DPMS "call real-mode procedure (IRET return)" calling convention:**

**Registers at call:**
AX = 0102h
CX = number of words of stack to copy
ES:DI -> callup/down register structure (see Table 31-21)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 31-20)

---

**31-5 DPMS "call real-mode interrupt handler" calling convention:**

**Registers at call:**
AX = 0103h
BL = interrupt number
CX = number of words of stack to copy
ES:DI -> callup/down register structure (see Table 31-21)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 31-20)

---

**31-6 DPMS "allocate descriptors" calling convention:**

**Registers at call:**
AX = 0200h
CX = number of descriptors to allocate

**Return Registers:**
CF clear if successful
AX = first descriptor allocated
CF set on error
AX = error code (see Table 31-20)

---

**31-7 DPMS "free a descriptor" calling convention:**

**Registers at call:**
AX = 0201h
BX = descriptor

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 31-20)

---

**31-8 DPMS "create alias descriptor" calling convention:**

**Registers at call:**
AX = 0202h
BX = descriptor

**Return Registers:**
CF clear if successful
AX = alias descriptor
CF set on error
AX = error code (see Table 31-20)

---

**31-9 DPMS "build alias to real-mode segment" calling convention:**

**Registers at call:**
AX = 0203h
BX = descriptor
CX = real-mode segment

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 31-20)

---

**31-10 DPMS "set descriptor base" calling convention:**

**Registers at call:**
AX = 0204h
BX = descriptor
CX:DX = base address

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 31-20)

---

**31-11 DPMS "set descriptor limit" calling convention:**

**Registers at call:**
AX = 0205h
BX = descriptor
CX = limit

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 31-20)

## INT 2Fh Function 43E0h

**31-12 DPMS "set descriptor type/attribute" calling convention:**

**Registers at call:**
AX = 0206h
BX = descriptor
CL = type
CH = attribute

**Return Registers:**
- CF clear if successful
- CF set on error
  - AX = error code (see Table 31-20)

**31-13 DPMS "get descriptor base" calling convention:**

**Registers at call:**
AX = 0207h
BX = descriptor

**Return Registers:**
- CF clear if successful
  - CX:DX = base address
- CF set on error
  - AX = error code (see Table 31-20)

**31-14 DPMS "get size of largest free block of memory" calling convention:**

**Registers at call:**
AX = 0300h

**Return Registers:**
- CF clear if successful
  - BX:CX = size
- CF set on error
  - AX = error code (see Table 31-20)

**31-15 DPMS "allocate block of extended memory" calling convention:**

**Registers at call:**
AX = 0301h
BX:CX = size

**Return Registers:**
- CF clear if successful
  - BX:CX = base address
  - SI:DI = handle
- CF set on error
  - AX = error code (see Table 31-20)

**31-16 DPMS "free block of extended memory" calling convention:**

**Registers at call:**
AX = 0302h
SI:DI = handle

**Return Registers:**
- CF clear if successful
- CF set on error
  - AX = error code (see Table 31-20)

**31-17 DPMS "map linear memory" calling convention:**

**Registers at call:**
AX = 0303h
ES:[DI] = DDS

**Return Registers:**
- CF clear if successful
  - BX:CX = base address
  - SI:DI = handle
- CF set on error
  - AX = error code (see Table 31-20)

**31-18 DPMS "unmap linear memory" calling convention:**

**Registers at call:**
AX = 0304h
SI:DI = handle

**Return Registers:**
- CF clear if successful
- CF set on error
  - AX = error code (see Table 31-20)

**31-19 DPMS "relocate segment to extended memory" calling convention:**

**Registers at call:**
AX = 0400h
ES:SI = base address
CX = limit
BL = type

BH = attribute
DX = selector or 0000h

**Return Registers:**
CF clear if successful
AX = selector
BX:CX = new base address
SI:DI = handle
CF set on error
AX = error code (see Table 31-20)

### 31-20 DPMS error codes:

| | |
|---|---|
| 8000h | general error |
| 8001h | unsupported function |
| 8011h | descriptor unavailable |
| 8012h | linear memory unavailable |
| 8013h | physical memory unavailable |
| 8021h | invalid value |
| 8022h | invalid selector |
| 8023h | invalid handle |

### 31-21 Format of callup/down register structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | EDI |
| 04h | DWORD | ESI |
| 08h | DWORD | EBP |
| 0Ch | 4 BYTEs | reserved (0) |
| 10h | DWORD | EBX |
| 14h | DWORD | EDX |
| 18h | DWORD | ECX |
| 20h | DWORD | EAX |
| 24h | DWORD | EIP |
| 28h | WORD | CS |
| 2Ah | 2 BYTEs | reserved (0) |
| 2Ch | DWORD | EFLAGS |
| 30h | DWORD | ESP |
| 34h | WORD | SS |
| 36h | 2 BYTEs | reserved (0) |
| 38h | WORD | ES |
| 3Ah | 2 BYTEs | reserved (0) |
| 3Ch | WORD | DS |
| 3Eh | 2 BYTEs | reserved (0) |
| 40h | WORD | FS |
| 42h | 2 BYTEs | reserved (0) |
| 44h | WORD | GS |
| 46h | 2 BYTEs | reserved (0) |

## CHAPTER 32

# *Cloaking*

RM386 (RAM-MAN/386) is the memory manager included in Helix Software's Netroom.

The Cloaking specification it supports allows TSRs to reside and run in protected-mode extended memory while leaving only a very small (typically 1K or less) stub in real-mode memory.

---

### INT 2Ch
*RM386 v6.00*

### *CLOAKING - CALL PROTECTED-MODE PASSALONG CHAIN*
*Real Mode*

**Details:** When this interrupt is invoked in V86 mode, RM386 will invoke the first in a chain of protected-mode handlers, and will only pass execution to the V86-mode INT 2Ch handler if none of the handlers in the passalong chain handle the call instead. This is the method by which the real-mode stub of a cloaked application communicates with the protected-mode portion.

**See also:** INT 2Ch Function 0009h below

---

### INT 2Ch Function 0000h
*RM386 v6.00*

### *CLOAKING - ALLOCATE GDT SELECTOR*
*Protected Mode*

**Registers at call:**

AX = 0000h

EBX = base address

CL = access mode byte

CH = extended access mode byte (omit limit field)

EDX = segment limit

**Return Registers:**

CF clear if successful

AX = selector

CF set on error

AX = error code (see Table 32-1)

**Details:** This INT 2Ch interface is used by Netroom's DPMI.EXE v3.00.

**See also:** Functions 0001h, 0002h, 0003h, 0004h, and 0005h; Netroom INT 31h Function 57h (Chapter 41)

**32-1 Cloaking error codes:**

| | |
|---|---|
| 0001h | no more selectors |
| 0002h | not a GDT ring 0 selector |
| 0003h | invalid selector (out of range, not user selector) |
| 0004h | selector not allocated |

---

### INT 2Ch Function 0001h
*RM386 v6.00*

### *CLOAKING - FREE GDT SELECTOR*
*Protected Mode*

**Registers at call:**

AX = 0001h

SI = selector

**Return Registers:**

CF clear if successful

CF set on error

AX = error code (see Table 32-1)

**See also:** Function 0000h

---

### INT 2Ch Function 0002h
### *CLOAKING - SET SEGMENT BASE ADDRESS*

*RM386 v6.00*
*Protected Mode*

**Registers at call:**
AX = 0002h
SI = selector
EBX = new physical base addres

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 32-1)

**See also:** Functions 0000h, 0003h, and 0004h; DPMI INT 31h Function 0007h (Chapter 30)

---

### INT 2Ch Function 0003h
### *CLOAKING - SET SEGMENT LIMIT*

*RM386 v6.00*
*Protected Mode*

**Registers at call:**
AX = 0003h
SI = selector
EBX = new limit

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 32-1)

**See also:** Functions 0000h, 0002h, and 0004h; DPMI INT 31h Function 0008h (Chapter 30)

---

### INT 2Ch Function 0004h
### *CLOAKING - SET SEGMENT ACCESS MODE*

*RM386 v6.00*
*Protected Mode*

**Registers at call:**
AX = 0004h
SI = selector
CL = new access mode byte

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 32-1)

**See also:** Functions 0000h, 0002h, 0003h, and 0005h; DPMI INT 31h Function 0009h (Chapter 30)

---

### INT 2Ch Function 0005h
### *CLOAKING - SET SEGMENT EXTENDED ACCESS MODE*

*RM386 v6.00*
*Protected Mode*

**Registers at call:**
AX = 0005h
SI = selector
CL = new extended access mode byte
(limit field ignored)

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 32-1)

**See also:** Functions 0000h, 0002h, 0003h, and 0004h; DPMI INT 31h Function 0009h (Chapter 30)

---

### INT 2Ch Function 0006h
### *CLOAKING - GET PROTECTED-MODE INTERRUPT VECTOR*

*RM386 v6.00*
*Protected Mode*

**Registers at call:**
AX = 0006h
CL = vector

**Return Registers:**
CF clear
DX:EBX -> current interrupt handler

**See also:** Function 0007h, DPMI INT 31h Function 0204h (Chapter 30)

---

### INT 2Ch Function 0007h
### *CLOAKING - SET PROTECTED-MODE INTERRUPT VECTOR*

*RM386 v6.00*
*Protected Mode*

**Registers at call:**
AX = 0007h
CL = vector
DX:EBX -> new interrupt handler

**Return Registers:**
CF clear

**See also:** Function 0006h, DPMI INT 31h Function 0205h (Chapter 30)

---

### INT 2Ch Function 0008h
### *CLOAKING - GET PASSALONG ADDRESS*

*RM386 v6.00*
*Protected Mode*

**Registers at call:**
AX = 0008h

**Return Registers:**
CF clear
DX:EBX = current passalong address

**See also:** Functions 0009h and 002Ch

## INT 2Ch Function 0009h

*RM386 v6.00*

### *CLOAKING - SET PASSALONG ADDRESS*

*Protected Mode*

**Registers at call:**
AX = 0009h
DX:EBX = new value for passalong address

**Return Registers:**
CF clear

**Details:** When an INT 2Ch instruction is executed in V86 mode, RM386 calls the passalong address. The handler should check whether the upcall is of interest to it, and if not it should jump to the old passalong address (retrieved with Function 0008h before the handler was installed).

**See also:** Functions 0008h and 002Dh, Cloaking INT 2Ch above

### 32-2 Passalong address calling convention:

**Registers at call:**
EAX = CS:IP of byte following INT 2Ch instruction invoking passalong
SS:EBX -> caller registers (see Table 32-3)
CF clear
others undefined

**Return Registers:**
CF clear: pass along to V86-mode INT 2Ch handler
CF set: return immediately to V86 mode

### 32-3 Format of caller registers:

| *Offset* | *Size* | *Description* | *Offset* | *Size* | *Description* |
|---|---|---|---|---|---|
| 00h | DWORD | EDI | 30h | DWORD | ESP |
| 04h | DWORD | ESI | 34h | WORD | SS |
| 08h | DWORD | EBP | 36h | WORD | padding |
| 0Ch | DWORD | reserved (ESP from PUSHAD) | **---rest not available if ring3 trap---** | | |
| 10h | DWORD | EBX | 38h | WORD | ES |
| 14h | DWORD | EDX | 3Ah | WORD | padding |
| 18h | DWORD | ECX | 3Ch | WORD | DS |
| 1Ch | DWORD | EAX | 3Eh | WORD | padding |
| 20h | DWORD | error code | 40h | WORD | FS |
| 24h | DWORD | EIP | 42h | WORD | padding |
| 28h | WORD | CS | 44h | WORD | GS |
| 2Ah | WORD | padding | 46h | WORD | padding |
| 2Ch | DWORD | EFLAGS | | | |

## INT 2Ch Function 000Ah

*RM386 v6.00*

### *CLOAKING - GET BASE ADDRESS OF GDT SELECTOR*

*Protected Mode*

**Registers at call:**
AX = 000Ah
SI = selector

**Return Registers:**
CF clear if successful
EBX = segment base address
CF set on error
AX = error code (see Table 32-1)

**See also:** Functions 0000h, 0002h, and 000Bh

## INT 2Ch Function 000Bh

*RM386 v6.00*

### *CLOAKING - GET SELECTOR LIMIT*

*Protected Mode*

**Registers at call:**
AX = 000Bh
SI = selector

**Return Registers:**
CF clear if successful
EBX = segment base address
CF set on error
AX = error code (see Table 32-1)

**See also:** Function 000Ah

---

**INT 2Ch Functions 000Ch to 001Fh** *RM386 v6.00*
***CLOAKING - RESERVED FOR CLOAKED BIOS USE UNDER WINDOWS*** *Protected Mode*
**Registers at call:**
AX = function (000Ch-001Fh)

---

**INT 2Ch Function 0020h** *RM386 v6.00*
***CLOAKING - GET SIZE OF PROTECTED-MODE STATE*** *Protected Mode*
**Registers at call:**
AX = 0020h
**Return Registers:**
EAX = number of bytes required for storing state
**See also:** Functions 0021h and 0022h

---

**INT 2Ch Function 0021h** *RM386 v6.00*
***CLOAKING - SAVE PROTECTED-MODE STATE*** *Protected Mode*
**Registers at call:**
AX = 0021h
ES:EDI -> buffer for protected-mode state
**Return Registers:**
CF clear
buffer filled
**See also:** Functions 0020h and 0022h

---

**INT 2Ch Function 0022h** *RM386 v6.00*
***CLOAKING - RESTORE PROTECTED-MODE STATE*** *Protected Mode*
**Registers at call:**
AX = 0022h
DS:ESI -> buffer containing
  previously-saved protected-mode state
**Return Registers:**
CF clear if successful
  state restored
CF set on error (invalid buffer contents)
**See also:** Functions 0020h and 0021h

---

**INT 2Ch Function 0023h** *RM386 v6.00*
***CLOAKING - ISSUE PROTECTED-MODE XMS CALL*** *Protected Mode*
**Registers at call:**
AX = 0023h
**Details:** This function is currently not implemented--it is a NOP in RM386 v6.00.

---

**INT 2Ch Function 0024h** *RM386 v6.00*
***CLOAKING - SET V86-MODE STACK*** *Protected Mode*
**Registers at call:**
AX = 0024h
DX:EBX = new value for V86-mode SS:ESP
**Return Registers:**
none
**See also:** Functions 0025h to 0027h

---

**INT 2Ch Function 0025h** *RM386 v6.00*
***CLOAKING - CALL V86-MODE PROCEDURE*** *Protected Mode*
**Registers at call:**
AX = 0025h
DS:EBX -> client register structure
  (see Function 0009h)
**Return Registers:**
CF clear if successful
  client register structure updated
CF set if no more nested procedure call space available

**Details:** This call uses the V86-mode stack supplied in the client structure, and calls the routine specified by CS:IP in the client structure.
**See also:** Functions 0026h and 0027h, DPMI INT 31h Function 0301h (Chapter 30)

---

**INT 2Ch Function 0026h** *RM386 v6.00*
***CLOAKING - CALL V86-MODE INTERRUPT HANDLER*** *Protected Mode*
**Registers at call:**
AX = 0026h
DS:EBX -> client register structure

(see Function 0009h)
CX = interrupt number

**Return Registers:**
CF clear if successful
client register structure updated
CF set if no more nested procedure call space available

**Details:** This call uses the V86-mode stack supplied in the client structure.
**See also:** Functions 0025h and 0027h, DPMI INT 31h Function 0300h (Chapter 30)

---

### INT 2Ch Function 0027h
### *CLOAKING - CHAIN TO V86-MODE INTERRUPT HANDLER*

*RM386 v6.00*
*Protected Mode*

**Registers at call:**
AX = 0027h
DS:EBX -> client register structure
(see Function 0009h)

**Return Registers:**
CF clear if successful
client register structure updated
CF set if no more nested procedure call space available

**Details:** This call uses the V86-mode stack supplied in the client structure, and jumps to the address specified by CS:IP in the client structure.
**See also:** Functions 0025h and 0026h

---

### INT 2Ch Function 0028h
### *CLOAKING - GET ESP0 FROM TSS*

*RM386 v6.00*
*Protected Mode*

**Registers at call:**
AX = 0028h

**Return Registers:**
CF clear
EAX = TSS's ESP0

---

### INT 2Ch Function 0029h
### *CLOAKING - SET SECONDARY STACK*

*RM386 v6.00*
*Protected Mode*

**Purpose:** Inform RM386 of the ring 3 interrupt stack location.

**Registers at call:**
AX = 0029h
DX:EBX = new value for SS:ESP of ring 3 secondary stack

**Return Registers:**
CF clear

---

### INT 2Ch Function 002Ah
### *CLOAKING - SET 8259 IRQ BASE VECTORS*

*RM386 v6.00*
*Protected Mode*

**Purpose:** Inform RM386 that the caller has changed the interrupt mappings.

**Registers at call:**
AX = 002Ah
BL = base vector of master interrupt controller
CL = base vector of slave interrupt controller

**See also:** VCPI INT 67h Function DE0Bh (Chapter 29)

---

### INT 2Ch Function 002Bh
### *CLOAKING - PROTECTED-MODE VIRTUAL DMA SERVICES*

*RM386 v6.00*
*Protected Mode*

**Registers at call:**
AX = 002Bh
CH = 81h
CL = subfunction (02h-0Ch)
other registers as appropriate for subfunction

**Return Registers:**
varies by function
CF set on error

**Details:** These functions are equivalent to the INT 4Bh Function 81xxh subfunctions with the same numbers. **See also:** VDS INT 4Bh Functions 8102h and 810Ch (Chapter 18)

---

### INT 2Ch Function 002Ch
### *CLOAKING - GET PORT-TRAPPING PASSALONG*

*RM386 v6.00*
*Protected Mode*

**Registers at call:**
AX = 002Ch

**Return Registers:**
CF clear
DX:EBX = current I/O trapping passalong address

**See also:** Functions 0008h and 002Dh

### INT 2Ch Function 002Dh
### *CLOAKING - SET PORT-TRAPPING PASSALONG*

*RM386 v6.00*
*Protected Mode*

**Registers at call:**
AX = 002Dh
DX:EBX = new I/O trapping passalong address

**Return Registers:**
CF clear

**Details:** RM386 calls the passalong address whenever an access to a monitored I/O port is attempted; the handler should check whether it is a port that it is interested in, and if not call the previous passalong address (which was retrieved with Function 002Ch before installing the new handler).

**See also:** Functions 0009h and 002Ch, RM386 INT 67h Function 5DEAh (Chapter 41)

#### 32-4 Port-trapping passalong address calling convention:

**Registers at call:**
EAX = CS:IP of faulting instruction (unless executing in protected-mode ring 3)
SS:EBX -> caller register structure (see Function 0009h); check EFLAGS V86-mode bit for type
CX = first two bytes of I/O instruction which was trapped
DX = port to which I/O is being performed
CF clear

**Return Registers:**
CF clear if RM386 should perform I/O operation
CF set if I/O should be skipped

**Details:** RM386 skips the trapped I/O instruction, so the passalong handler should not modify the client CS:EIP.

### INT 2Ch Function 002Eh
### *CLOAKING - TRAP I/O PORT*

*RM386 v6.00*
*Protected Mode*

**Registers at call:**
AX = 002Eh
DX = port number to trap

**Return Registers:**
CF clear if successful
CF set on error (port out of range or reserved)

**See also:** Functions 002Fh and 0030h

### INT 2Ch Function 002Fh
### *CLOAKING - UNTRAP I/O PORT*

*RM386 v6.00*
*Protected Mode (undocumented)*

**Registers at call:**
AX = 002Fh
DX = port number for which to cancel trapping

**Return Registers:**
CF clear if successful
CF set on error (port out of range or reserved)

**See also:** Functions 002Eh and 0030h

### INT 2Ch Function 0030h
### *CLOAKING - GET TRAPPING STATE OF PORT*

*RM386 v6.00*
*Protected Mode (undocumented)*

**Registers at call:**
AX = 0030h
DX = port number

**Return Registers:**
CF clear if successful BX = current state (0000h not trapped, 0001h trapped)
CF set on error (port out of range or reserved)

**See also:** Functions 002Eh and 002Fh

### INT 2Ch Function 0031h
### *BUG*

*RM386 v6.00*
*Protected Mode (undocumented)*

**Registers at call:**
AX = 0031h

**Return Registers:**
crash

**Details:** Due to a fencepost error, RM386 v6.00 will branch unpredictably if invoked with this function

## CHAPTER 33

# *Phar Lap and FlashTek DOS Extenders*

Phar Lap Software was one of the four principal players when DOS extenders first came on the scene back in 1987 and 1988, and has remained in the forefront of the technology ever since. FlashTek's "DOSX" 32-bit extender first shipped with the Zortech C/C++ compiler in July, 1991, and became a separate product under the FlashTek name when Zortech was acquired by Symantec. This chapter describes the interrupts used by both product lines, since the FlashTek API duplicates large parts of the Phar Lap functionality for compatibility.

## Phar Lap

Phar Lap Software has been a major supplier of 386 DOS development tools from the beginning, and defined the 386 object module (EASYOMF) format commonly used under DOS. With few exceptions, most 386 DOS compilers can generate Phar Lap 386 object files and provide Phar Lap-compatible libraries.

The Phar Lap DOS extender can use memory from all available memory APIs, and Phar Lap was the first vendor to offer a virtual-memory option for 386 applications.

---

### INT 21h Function 2501h — *Phar Lap 386/DOS-Extender*
### *RESET DOS EXTENDER DATA STRUCTURES* — *Protected Mode*

| Registers at call: | Return Registers: |
|---|---|
| AX = 2501h | CF clear if successful |
| SS = application's original SS or DS (FlashTek X-32VM) | CF set on error<br>caller is operating on X-32 stack (FlashTek X-32VM) |

**Details:** Phar Lap uses INT 21h Function 25h as the entry point for all 386/DOS-Extender system calls. They are only available when directly using 386/DOS-Extender or a compatible DOS extender, or when using a product that was created using 386-DOS/Extender or a compatible.

This function is also supported by FlashTek X-32VM.

**See also:** Phar Lap Function 30h

---

### INT 21h Function 2502h — *Phar Lap 386/DOS-Extender*
### *GET PROTECTED-MODE INTERRUPT VECTOR* — *Protected Mode*

| Registers at call: | Return Registers: |
|---|---|
| AX = 2502h | CF clear |
| CL = interrupt number | ES:EBX = CS:EIP of protected-mode interrupt handler |

**Details:** This function is also supported by FlashTek X-32VM.

**See also:** Functions 2503h and 2504h, DPMI INT 31h Function 0204h (Chapter 30)

---

### INT 21h Function 2503h

### *GET REAL-MODE INTERRUPT VECTOR*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2503h
CL = interrupt number

**Return Registers:**
CF clear
EBX = CS:IP of real-mode interrupt handler

**Details:** This function is also supported by FlashTek X-32VM.

**See also:** Functions 2502h and 2504h, MS-DOS Function 35h (Chapter 24), DPMI INT 31h Function 0200h (Chapter 30)

---

### INT 21h Function 2504h

### *SET PROTECTED-MODE INTERRUPT VECTOR*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2504h
CL = interrupt number
DS:EDX = CS:EIP of protected-mode interrupt handler

**Return Registers:**
CF clear

**Details:** This function is also supported by FlashTek X-32VM.

**See also:** Functions 2502h and 2505h, DPMI INT 31h Function 0205h (Chapter 30)

---

### INT 21h Function 2505h

### *SET REAL-MODE INTERRUPT VECTOR*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2505h
CL = interrupt number
EBX = CS:IP of real-mode interrupt handler

**Return Registers:**
CF clear

**Details:** This function is also supported by FlashTek X-32VM.

**See also:** Functions 2503h and 2504h, DPMI INT 31h Function 0201h (Chapter 30)

---

### INT 21h Function 2506h

### *SET INT TO ALWAYS GAIN CONTROL IN PROTECTED MODE*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2506h
CL = interrupt number
DS:EDX = CS:EIP of protected-mode interrupt handler

**Return Registers:**
CF clear

**Details:** This function modifies both the real-mode low-memory interrupt vector table and the protected-mode Interrupt Descriptor Table (IDT). Interrupts occurring in real mode are resignaled in protected mode. This function is also supported by FlashTek X-32VM.

---

### INT 21h Function 2507h

### *SET REAL- AND PROTECTED-MODE INTERRUPT VECTORS*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2507h
CL = interrupt numbern
DS:EDX = CS:EIP of protected-mode interrupt handler
EBX = CS:IP of real-mode interrupt handler

**Return Registers:**
CF clear

**Details:** Interrupts are disabled until both vectors have been modified.

This function is also supported by FlashTek X-32VM.

**See also:** Functions 2504h and 2505h

---

### INT 21h Function 2508h

### *GET SEGMENT LINEAR BASE ADDRESS*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2508h
BX = segment selector

**Return Registers:**
CF clear if successful

ECX = linear base address of segment
CF set if invalid segment selector

**Details:** This function is also supported by FlashTek X-32VM.

**See also:** Function 2509h, DPMI INT 31h Function 0006h (Chapter 30)

---

### INT 21h Function 2509h
### *CONVERT LINEAR TO PHYSICAL ADDRESS*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2509h
EBX = linear address to convert

**Return Registers:**
CF clear if successful
ECX = physical address (carry flag clear)
CF set if linear address not mapped in page tables

**See also:** Function 2508h

---

### INT 21h Function 250Ah
### *MAP PHYSICAL MEMORY AT END OF SEGMENT*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 250Ah
ES = Local Descriptor Table (LDT) selector of segment to modify
EBX = physical base address of memory to map (multiple of 4K)
ECX = number of physical 4K pages to map

**Return Registers:**
CF clear if successful
EAX = 32-bit offset in segment of mapped memory
CF set on error
EAX = error code (08h,09h) (see Table 33-1)

**See also:** DPMI INT 31h Function 0800h (Chapter 30)

**33-1 Phar Lap DOS Extender error codes:**

| | |
|---|---|
| 01h | not enough real-mode stack space |
| 02h | file not found or path invalid |
| 05h | access denied |
| 07h | MS-DOS memory control blocks destroyed |
| 08h | insufficient memory for requested operation |
| 09h | invalid segment selector, invalid address range, or invalid address |
| 0Ah | environment invalid |
| 0Bh | invalid file format |
| 80h | LDT too small |
| 81h | invalid parameters or 386|VMM not present |
| 82h | running under DPMI |
| 86h | all 386|VMM file handles already in use |

---

### INT 21h Function 250Ch
### *GET HARDWARE INTERRUPT VECTORS*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 250Ch

**Return Registers:**
CF clear
AL = base interrupt vector for IRQ0-IRQ7
AH = base interrupt vector for IRQ8-IRQ15
BL = interrupt vector for BIOS print screen function (Phar Lap only)

**Details:** This function is also supported by FlashTek X-32VM.

**See also:** DPMI INT 31h Function 0400h (Chapter 30), VCPI INT 67h Function DE0Ah (Chapter 29)

## INT 21h Function 250Dh
### GET REAL-MODE LINK INFORMATION

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 250Dh

**Return Registers:**
CF clear
EAX = CS:IP of real-mode callback procedure that will call through from real mode to a protected-mode routine (see Table 33-2)
EBX = 32-bit real-mode address of intermode call data buffer
ECX = size in bytes of intermode call data buffer
ES:EDX = protected-mode address of intermode call data buffer

**Details:** This function is also supported by FlashTek X-32VM.

X-32VM guarantees the intermode buffer to be at least 4 KB.

**See also:** Function 250Eh

### 33-2 real-mode callback calling convention:

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| STACK: | DWORD | offset to protected-mode code | via FAR return |
| | WORD | placeholder for protected-mode CS | |
| | DWORD | pointer to selector structure (see Table 33-3) or 0000h:0000h for defaults | |
| | var | parameters for protected-mode procedure | |

### 33-3 Format of selector structure:

| Offset | Size | Description |
|---|---|---|
| 00h | WORD | protected-mode GS selector |
| 02h | WORD | protected-mode FS selector |
| 04h | WORD | protected-mode ES selector |
| 06h | WORD | protected-mode DS selector |

## INT 21h Function 250Eh
### CALL REAL-MODE PROCEDURE

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 250Eh
EBX = CS:IP of real-mode procedure to call
ECX = number of two-byte words to copy from protected-mode stack to real-mode stack

**Return Registers:**
CF clear if successful
- all segment registers unchanged
- all general registers contain values set by real-mode procedure
- all other flags set as they were left by real-mode procedure
- stack unchanged

CF set on error
- EAX = error code (01h) (see Table 33-1)

**Details:** This function is also supported by FlashTek X-32VM; under X-32VM, the call will fail if ECX > 0000003Fh.

**See also:** Functions 250Dh and 2510h, OS/286 Function E1h (Chapter 34), DPMI INT 31h Function 0301h (Chapter 30)

## INT 21h Function 250Fh
### CONVERT PROTECTED-MODE ADDRESS TO MS-DOS

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 250Fh
ES:EBX = 48-bit protected-mode address to convert

**Return Registers:**
CF clear if successful (address < 1MB)
- ECX = 32-bit real-mode MS-DOS address

CF set on error (address >= 1MB)
- ECX = linear address

**Details:** This function is also supported by FlashTek X-32VM.
**See also:** Function 2510h

---

## INT 21h Function 2510h
## *CALL REAL-MODE PROCEDURE, USING REGISTERS*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2510h
EBX = CS:IP of real-mode procedure to call
ECX = number of two-byte words to copy to protected-mode stack to real-mode stack
DS:EDX -> pointer to parameter block (see Table 33-4)

**Return Registers:**
CF clear if successful
- all segment registers unchanged
- EDX unchanged
- all other general registers contain values set by real-mode procedure
- all other flags are set as they were left by real-mode procedure
- real-mode register values are returned in the parameter block

CF set on error
- EAX = error code (01h) (see Table 33-1)

**Details:** Unlike most of the preceding 25xxh functions, this one is not supported by FlashTek X-32VM.
**See also:** Functions 250Eh and 250Fh

**33-4 Format of parameter block:**

| *Offset* | *Size* | *Description* | *Offset* | *Size* | *Description* |
|---|---|---|---|---|---|
| 00h | WORD | real-mode DS value | 08h | DWORD | real-mode EAX value |
| 02h | WORD | real-mode ES value | 0Ch | DWORD | real-mode EBX value |
| 04h | WORD | real-mode FS value | 10h | DWORD | real-mode ECX value |
| 06h | WORD | real-mode GS value | 14h | DWORD | real-mode EDX value |

---

## INT 21h Function 2511h
## *ISSUE REAL-MODE INTERRUPT*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2511h
DS:EDX -> parameter block (see Table 33-5)

**Return Registers:**
all segment registers unchanged
EDX unchanged
all other registers contain values set by the real-mode interrupt handler
the flags are set as they were left by the real-mode interrupt handler
real-mode register values are returned in the parameter block

**Details:** This function is also supported by FlashTek X-32VM.
**See also:** Functions 2503h, 2505h, and 250Eh; OS/286 Function E3h (Chapter 34); DPMI INT 31h Function 0300h (Chapter 30)

**33-5 Format of parameter block:**

| *Offset* | *Size* | *Description* | *Offset* | *Size* | *Description* |
|---|---|---|---|---|---|
| 00h | WORD | interrupt number | 08h | WORD | real-mode GS value |
| 02h | WORD | real-mode DS value | 0Ah | DWORD | real-mode EAX value |
| 04h | WORD | real-mode ES value | 0Eh | DWORD | real-mode EDX value |
| 06h | WORD | real-mode FS value | | | |

**Note:** All other real-mode values are set from protected-mode registers.

### INT 21h Function 2512h
### *LOAD PROGRAM FOR DEBUGGING*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2512h
DS:EDX -> pointer to ASCIZ program name
ES:EBX -> pointer to parameter block (see Table 33-6)
ECX = size in bytes of LDT buffer

**Return Registers:**
CF clear if successful
EAX = number of segment descriptors in LDT
CF set on error
EAX = error code (02h,05h,08h,0Ah,0Bh,80h) (see Table 33-1)

**33-6 Format of parameter block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| **Input:** | | |
| 00h | DWORD | 32-bit offset of environment string |
| 04h | WORD | segment of environment string |
| 06h | DWORD | 32-bit offset of command-tail string |
| 0Ah | WORD | segment of command-tail string |
| 0Ch | DWORD | 32-bit offset of LDT buffer (size in ECX) |
| 10h | WORD | segment of LDT buffer |
| **Output:** | | |
| 12h | WORD | real-mode paragraph address of PSP (see also Table 24-!M) |
| 14h | WORD | real/protected mode flag (0000h real mode, 0001h protected mode) |
| 16h | DWORD | initial EIP value |
| 1Ah | WORD | initial CS value |
| 1Ch | DWORD | initial ESP value |
| 20h | WORD | initial SS value |
| 22h | WORD | initial DS value |
| 24h | WORD | initial ES value |
| 26h | WORD | initial FS value |
| 28h | WORD | initial GS value |

### INT 21h Function 2513h
### *ALIAS SEGMENT DESCRIPTOR*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2513h
BX = segment selector of descriptor in GDT or LDT
CL = access-rights byte for alias descriptor
CH = use-type bit (USE16 or USE32) for alias descriptor

**See also:** Function 2514h

**Return Registers:**
CF clear if successful
AX = segment selector for created alias
CF set on error
EAX = error code (08h,09h) (see Table 33-1)

### INT 21h Function 2514h
### *CHANGE SEGMENT ATTRIBUTES*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2514h
BX = segment selector of descriptor in GDT or LDT
CL = new access-rights byte
CH = new use-type bit (USE16 or USE32)

**Return Registers:**
CF clear if successful
CF set on error
EAX = error code (09h) (see Table 33-1)

**See also:** Function 2515h, DPMI INT 31h Function 0009h (Chapter 30)

---

### INT 21h Function 2515h
### *GET SEGMENT ATTRIBUTES*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2515h
BX = segment selector of descriptor in GDT or LDT

**Return Registers:**
CF clear if successful
  CL = access-rights byte for segment
  CH = use-type bit (USE16 or USE32)
ECX bits 16-31 destroyed
CF set on error
  EAX = error code (09h) (see Table 33-1)

**See also:** Function 2514h

---

### INT 21h Function 2516h
### *FREE ALL MEMORY OWNED BY LDT*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2516h
*others, if any, unknown*

**Return Registers:**
*unknown*

---

### INT 21h Function 2517h
### *GET INFORMATION ON DOS DATA BUFFER*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2517h
*others, if any, unknown*

**Return Registers:**
*unknown*

---

### INT 21h Function 2518h
### *SPECIFY HANDLER FOR MOVED SEGMENTS*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2518h
*others, if any, unknown*

**Return Registers:**
*unknown*

---

### INT 21h Function 2519h
### *GET ADDITIONAL MEMORY ERROR INFORMATION*

*Phar Lap 386/DOS-Extender VMM*
*Protected Mode*

**Registers at call:**
AX = 2519h

**Return Registers:**
CF clear
EAX = error code (see Table 33-7)

**Details:** VMM is the Virtual Memory Manager option.

**33-7 Phar Lap VMM error codes:**

| | |
|---|---|
| 0000h | no error |
| 0001h | out of physical memory |
| 0002h | out of swap space (unable to grow swap file) |
| 0003h | out of LDT entries and unable to grow LDT |
| 0004h | unable to change extended memory allocation mark |
| FFFFFFFFh | paging disabled |

---

### INT 21h Function 251Ah
### *LOCK PAGES IN MEMORY*

*Phar Lap 386/DOS-Extender VMM*
*Protected Mode*

**Registers at call:**
AX = 251Ah
EDX = number of 4k pages to lock
if BL = 00h
  ECX = linear address of first page to lock
if BL = 01h
  ES:ECX -> pointer to first page to lock

**Return Registers:**
CF clear if successful
CF set on error
  EAX = error code (08h,09h) (see also Table 33-1)

**See also:** Function 251Bh, OS/386 Function EB06h (Chapter 34), DPMI INT 31h Function 0600h (Chapter 30)

---

### INT 21h Function 251Bh
### *UNLOCK PAGES*

*Phar Lap 386/DOS-Extender VMM*
*Protected Mode*

**Registers at call:**
AX = 251Bh
EDX = number of pages to unlock
if BL = 00h
    ECX = linear address of first page to unlock
if BL = 01h
    ES:ECX -> pointer to first page to unlock

**Return Registers:**
CF clear if successful
CF set on error
    EAX = error code (09h) (see Table 33-1)

**See also:** Function 251Ah, OS/386 Function EB07h (Chapter 34), DPMI INT 31h Function 0601h (Chapter 30)

---

### INT 21h Function 251Dh
### *READ PAGE-TABLE ENTRY*

*Phar Lap 386/DOS-Extender VMM*
*Protected Mode*

**Registers at call:**
AX = 251Dh
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Functions 251Eh and 251Fh, OS/386 Function EB00h (Chapter 34), DPMI INT 31h Function 0506h (Chapter 30)

---

### INT 21h Function 251Eh
### *WRITE PAGE-TABLE ENTRY*

*Phar Lap 386/DOS-Extender VMM*
*Protected Mode*

**Registers at call:**
AX = 251Eh
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Functions 251Dh and 251Fh, DPMI INT 31h Function 0507h (Chapter 30)

---

### INT 21h Function 251Fh
### *EXHANGE TWO PAGE-TABLE ENTRIES*

*Phar Lap 386/DOS-Extender VMM*
*Protected Mode*

**Registers at call:**
AX = 251Fh
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Functions 251Dh and 251Eh

---

### INT 21h Function 2520h
### *GET MEMORY STATISTICS*

*Phar Lap 386/DOS-Extender VMM*
*Protected Mode*

**Registers at call:**
AX = 2520h
DS:EDX -> pointer to buffer at least
  100 bytes in size (see Table 33-8)
BL = reset flag
    00h don't reset VM stats, 01h reset stats

**Return Registers:**
carry flag clear

**33-8 Format of VM stats buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | VM status (0001h VM subsystem is present, 0000h VM not present) |
| 04h | DWORD | "nconvpg" number of conventional memory pages available |
| 08h | DWORD | "nbimpg" number of Compaq built-in memory pages available |
| 0Ch | DWORD | "nextpg" total number of extended memory pages |
| 10h | DWORD | "extlim" extender memory pages limit |
| 14h | DWORD | "aphyspg" number of physical memory pages allocated to appl |

| | | |
|---|---|---|
| 18h | DWORD | "alockpg" number of locked pages owned by application |
| 1Ch | DWORD | "sysphyspg" number physical memory pages allocated to system |
| 20h | DWORD | "nfreepg" number of free physical pages; approx if EMS VCPI |
| 24h | DWORD | linear address of beginning of application address space |
| 28h | DWORD | linear address of end of application address space |
| 2Ch | DWORD | number of seconds since last time VM stats were reset |
| 30h | DWORD | number of page faults since last time |
| 34h | DWORD | number of pages written to swap file since last time |
| 38h | DWORD | number of reclaimed pages (page faults on swapped pages) |
| 3Ch | DWORD | number of virtual pages allocated to the application |
| 40h | DWORD | size in pages of swap file |
| 44h | DWORD | number of system pages allocated with EMS calls |
| 48h | DWORD | minimum number of conventional memory pages |
| 4Ch | DWORD | maximum size in bytes to which swap file can be increased |
| 50h | DWORD | "vmflags" bit 0 = 1 if page fault in progress |
| 54h | 16 BYTEs | reserved for future expansion (set to zero) |

---

## INT 21h Function 2521h
### *LIMIT PROGRAM'S EXTENDED MEMORY USAGE*

*Phar Lap 386/DOS-Extender VMM*
*Protected Mode*

**Registers at call:**
AX = 2521h
EBX = max 4k pages of physical extended memory which program may use

**Return Registers:**
CF clear if successful
EBX = maximum limit in pages
ECX = minimum limit in pages
CF set on error
EAX = error code
08h insufficient memory or -nopage switch used

**See also:** Function 2522h

---

## INT 21h Function 2522h
### *SPECIFY ALTERNATE PAGE-FAULT HANDLER*

*Phar Lap 386/DOS-Extender VMM*
*Protected Mode*

**Registers at call:**
AX = 2522h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 2523h

---

## INT 21h Function 2523h
### *SPECIFY OUT-OF-SWAP-SPACE HANDLER*

*Phar Lap 386/DOS-Extender VMM*
*Protected Mode*

**Registers at call:**
AX = 2523h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 2522h

---

## INT 21h Function 2524h
### *INSTALL PAGE-REPLACEMENT HANDLERS*

*Phar Lap 386/DOS-Extender VMM*
*Protected Mode*

**Registers at call:**
AX = 2524h
*others, if any, unknown*

**Return Registers:**
*unknown*

---

## INT 21h Function 2525h
### *LIMIT PROGRAM'S CONVENTIONAL MEMORY USAGE*

*Phar Lap 386/DOS-Extender VMM*
*Protected Mode*

**Registers at call:**
AX = 2525h
EBX = limit in 4k pages of physical

conventional memory which
program may use

**Return Registers:**
CF clear if successful
EBX = maximum limit in pages
ECX = minimum limit in pages
CF set on error
EAX = error code
08h insufficient memory or -nopage switch used

**See also:** Function 2521h

---

### INT 21h Function 2526h
### *GET CONFIGURATION INFORMATION*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2526h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Details were not available at the time of writing.

**33-9 Format of configuration buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | flags 1<br>bit 24: set if -NESTDPMI specified<br>bit 25: set if -NONESTDPMI specified<br>bit 26: set if -NODPMI specified<br>bit 27: set if -NOPCDWEITEK specified |

---

### INT 21h Function 2529h
### *LOAD FLAT MODEL .EXP or .REX FILE*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 2529h
*others, if any, unknown*

**Return Registers:**
ES:EBX -> parameter block (see Table 33-10)
*others, if any, unknown*

**Details:** Details were not available at the time of writing.

**33-10 Format of parameter block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 28 BYTEs | *unknown* |
| 1Ch | DWORD | flags<br>bit 0: child linked with -UNPRIVILEGED<br>bits 1-31 reserved |

---

### INT 21h Function 252Bh, Subfunction 09h
### *GET PAGE TABLE ENTRY/PAGE TABLE INFORMATION*

*Phar Lap 386/DOS-Extender v4.1*
*Protected Mode*

**Registers at call:**
AX = 252Bh
BH = 09h
BL = subfunction
00h get page table entry by linear address
ECX = linear address for which to get page table entry
01h get page table entry by logical address
ES:ECX = address for which to get page table entry

**Return Registers:**
CF clear if successful
EAX = page table entry
EBX = additional page table information
CF set on error
EAX = error code (09h,82h) (see Table 33-1)

**See also:** Function 252Bh Subfunction 0Ah

---

### INT 21h Function 252Bh, Subfunction 0Ah
Phar Lap 386/DOS-Extender v4.1

### SET PAGE TABLE ENTRY/PAGE TABLE INFO
Protected Mode

**Registers at call:**
AX = 252Bh
BH = 0Ah
BL = subfunction
    00h set page table entry for linear address
        ECX = linear address for which to get page table entry
    01h set page table entry for logical address
        ES:ECX = address for which to get page table entry
ESI = page table entry
EDI = additional page table information

**Return Registers:**
CF clear if successful
CF set on error
    EAX = error code (09h,82h) (see Table 33-1)

**See also:** Function 252Bh Subfunction 09h

---

### INT 21h Function 252Bh, Subfunction 0Bh
Phar Lap 386/DOS-Extender v4.1

### MAP DATA FILE AT FILE OFFSET
Protected Mode

**Registers at call:**
AX = 252Bh
BH = 0Bh
BL = subfunction
    00h by linear address
        ECX = linear address at which to map data file
    01h by logical address
        ES:ECX = logical address at which to map data file
EDX = number of bytes to map
DS:ESI -> mapping structure (see Table 33-11)
DS:EDI -> ASCIZ filename

**Return Registers:**
CF clear if successful
CF set on error
    EAX = error code (02h,09h,81h,86h) (see also Table 33-1)
        0002h file error
            ECX = phase (01h opening file, 02h seeking, 03h reading)
            EDX = error code returned by DOS

**See also:** Function 252Bh Subfunction 09h

**33-11 Format of mapping structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | starting file offset to be mapped |
| 04h | DWORD | DOS file access and sharing modes (see Table 24-!L) |

---

### INT 21h Function 2534h
Phar Lap 386/DOS-Extender v4.0+

### GET INTERRUPT FLAG
Protected Mode

**Registers at call:**
AX = 2534h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Details were not available at the time of writing.

---

### INT 21h Function 2535h
Phar Lap 386/DOS-Extender v4.0+

### READ/WRITE SYSTEM REGISTERS
Protected Mode

**Registers at call:**
AX = 2535h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** This function is only available under MS Windows if PHARLAP.386 VDD is installed. Details were not available at the time of writing.

---

### INT 21h Function 253Ch
### *SHRINK 386|VMM SWAP FILE*

*Phar Lap 386/DOS-Extender v4.0+*
*Protected Mode*

**Registers at call:**
AX = 253Ch
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Details were not available at the time of writing.

---

### INT 21h Function 253Dh
### *READ/WRITE IDT DESCRIPTOR*

*Phar Lap 386/DOS-Extender v4.0+*
*Protected Mode*

**Purpose:** Access hardware-level IDT rather than the internal 386/DOS-Extender shadow IDT.

**Registers at call:**
AX = 253Dh
BL = interrupt number
ECX = direction (0 = read, 1 = write)
DS:EDX -> 8-byte buffer for descriptor

**Return Registers:**
CF clear if successful
    DS:EDX filled if reading
CF set on error
    EAX = error code (82h) (see Table 33-1)

**Details:** This call will always fail under DPMI because it is not possible to access the IDT.

The descriptor is not checked when writing.

This call can normally be used only by programs running in ring 0 because the processor does not allow an interrupt to be vectored to a less privileged ring.

---

### INT 21h Function 25C0h
### *ALLOCATE MS-DOS MEMORY BLOCK*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 25C0h
BX = number of 16-byte paragraphs of MS-DOS memory requested

**Return Registers:**
CF clear if successful
    AX = real-mode paragraph address of memory
CF set on error
    AX = error code (07h,08h) (see Table 33-1)
    BX = size in paragraphs of largest available memory block

**See also:** Functions 25C1h and 25C2h

---

### INT 21h Function 25C1h
### *RELEASE MS-DOS MEMORY BLOCK*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 25C1h
CX = real-mode paragraph address of memory block to free

**Return Registers:**
CF clear if successful
    EAX destroyed
CF set on error
    AX = error code (07h,09h) (see Table 33-1)

**See also:** Functions 25C0h and 25C2h

---

### INT 21h Function 25C2h
### *MODIFY MS-DOS MEMORY BLOCK*

*Phar Lap 386/DOS-Extender*
*Protected Mode*

**Registers at call:**
AX = 25C2h
BX = new requested block size in paragraphs
CX = real-mode paragraph address of memory block to modify

**Return Registers:**
CF clear if successful
    EAX destroyed
CF set on error
    AX = error code (07h,08h,09h) (see Table 33-1)
    BX = size in paragraphs of largest available memory block

**See also:** Functions 25C0h and 25C1h

---

**INT 21h Function 25C3h** *Phar Lap 386/DOS-Extender*

***EXECUTE PROGRAM*** *Protected Mode*

**Registers at call:**
AX = 25C3h
ES:EBX -> pointer to parameter block
(see Table 33-12)
DS:EDX -> pointer to ASCIZ
program filename

**Return Registers:**
CF clear if successful
all registers unchanged
CF set on error
EAX = error code (01h,02h,05h,08h,0Ah,0Bh)
(see also Table 33-1)
01h *function code in AL is invalid*

**33-12 Format of parameter block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | 32-bit offset of environment string |
| 04h | WORD | segment selector of environment string |
| 06h | DWORD | 32-bit offset of command-tail string |
| 0Ah | WORD | segment selector of command-tail string |

---

**INT 21h Function 30h** *Phar Lap 386/DOS-Extender, Intel Code Builder*

***INSTALLATION CHECK***

**Registers at call:**
AH = 30h
EAX = 00003000h
EBX = 50484152h ("PHAR")

**Return Registers:**
AL = major DOS version
AH = minor DOS version
EAX bits 31-16 = 4458h ('DX') if
386/DOS-extender installed
BL = ASCII major version number
EAX bits 31-16 = 4243h ('BC') if Intel Code
Builder installed
EDX = address of GDA

**See also:** Function 2501h, DOS/4G Function FF00h (Chapter 35), DOS Extender INT 2Fh Function F100h (Chapter 37)

---

**INT 2Fh Function 4040h** *PharLap 286|DOS-Extender Lite v2.5*

***UNDETERMINED FUNCTION***

**Registers at call:**
AX = 4040h

**Return Registers:**
BX:CX -> *unknown*

---

**INT 2Fh Function ED00h** *Phar Lap DOS EXTENDERS*

***INSTALLATION CHECK***

**Registers at call:**
AX = ED00h
BL = DOS extender
01h 286dosx v1.3+ (Software
Development Kit)
02h 286dosx v1.3+ (Run-Time Kit)
03h 386dosx v4.0+ (SDK)
04h 386dosx v4.0+ (RTK)

**Return Registers:**
AL = status
00h not installed
FFh installed
SI = 5048h ("PH")
DI = 4152h ("AR")
CH = major version number
CL = minor version number
DX = flags
bit 0: running under DPMI
bit 1: running under Phar Lap VMM

**Conflicts:** none, but see Table 2-2

**See also:** Ergo Function A1h (Chapter 34), DOS Extender Function F100h (Chapter 37), DPMILOAD Function FBA1h (Chapter 36)

### INT 2Fh Function ED03h

*Phar Lap 386/DOS-Extender v4.1*
*Real Mode*

### *GET EXTENDER ENTRY POINT*

**Registers at call:**
AX = ED03h
CX = real-mode code segment
DX = real-mode data segment

**Return Registers:**
CF clear if successful
CX = protected-mode code segment selector
DX = protected-mode data segment selector
ES:DI -> real-mode entry point for calling protected-mode functions
CF set on error
AX = error code
0008h unable to allocate LDT descriptors

**Conflicts:** none, but see Table 2-2

### INT 2Fh Function ED80h

*Phar Lap 286|DOS Extender Lite v2.5*

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = ED80h
BL = DOS extender ID (see Function ED00h)
SI = 5048h ("PH")
DI = 4152h ("AR")
*others, if any, unknown*

**Return Registers:**
*unknown*

**Conflicts:** none, but see Table 2-2

## FlashTek Extensions

The FlashTek X-32VM DOS extender was first distributed as a royalty-free DOS extender known as DOSX which was included as part of the Zortech C++ compiler. When Zortech was acquired by Symantec, this package was spun off as a separate product by its original developer, FlashTek Inc. This section describes only those parts of the FlashTek API that differ from the Phar Lap interface already listed.

### INT 21h Function 2509h

*FlashTek X-32VM*

### *GET SYSTEM SEGMENTS AND SELECTORS*

**Registers at call:**
AX = 2509h

**Return Registers:**
CF clear
EAX high word = default DS
AX = alias for 16-bit data segment
BX = real mode code segment
EDX high word = selector covering full 4GB address space
DX = default SS
ESI high word = PSP selector
SI = environment selector

### INT 21h Function 252Bh

*FlashTek X-32VM*

### *VIRTUAL MEMORY MANAGEMENT - PAGE LOCKING*

**Registers at call:**
AX = 252Bh
BH = function
05h lock pages
06h unlock pages

BL = address type
  00h linear address
    ECX = linear start address of memory region
  01h segmented address
    ES:ECX -> start of memory region
EDX = size of memory region in bytes

**Return Registers:**
CF clear if successful
CF set on error

**Details:** If X-32 is not using virtual memory, this function always succeeds.

---

### INT 21h Function 2532h
*FlashTek X-32VM*

### *GET EXCEPTION HANDLER VECTOR*

**Registers at call:**
AX = 2532h
CL = exception number (00h-0Fh)

**Return Registers:**
CF clear if successful
  ES:EBX = CS:EIP of current exception handler
CF set on error (CL > 0Fh)

**See also:** Function 2533h

---

### INT 21h Function 2533h
*FlashTek X-32VM*

### *SET EXCEPTION HANDLER VECTOR*

**Registers at call:**
AX = 2533h
CL = exception number (00h-0Fh)
DS:EDX = CS:EIP of new exception handler

**Return Registers:**
CF clear if successful
CF set on error (CL > 0Fh)

**See also:** Function 2532h

---

### INT 21h Function 3501h
*FlashTek X-32VM*
*Protected Mode*

### *ALLOCATE PROTECTED-MODE SELECTOR*

**Registers at call:**
AX = 3501h

**Return Registers:**
CF clear if successful
  BX = new selector
CF set on error (no more selectors available)

**Details:** The new selector will be an expand-up read/write data selector with undefined base and limit.

**See also:** Function 3502h, DPMI INT 31h Function 0000h (Chapter 30)

---

### INT 21h Function 3502h
*FlashTek X-32VM*
*Protected Mode*

### *DEALLOCATE PROTECTED-MODE SELECTOR*

**Registers at call:**
AX = 3502h
BX = selector

**Return Registers:**
CF clear if successful
CF set on error (invalid selector)

**Details:** Only selectors allocated via Function 3501h should be deallocated.

**See also:** Function 3501h, DPMI INT 31h Function 0001h (Chapter 30)

---

### INT 21h Function 3503h
*FlashTek X-32VM*
*Protected Mode*

### *SET SELECTOR BASE ADDRESS*

**Registers at call:**
AX = 3503h
BX = selector
ECX = base address

**Return Registers:**
CF clear if successful
CF set on error (invalid selector)

**See also:** Functions 3504h and 3505h, DPMI INT 31h Function 0007h (Chapter 30)

### INT 21h Function 3504h
### *GET SELECTOR BASE ADDRESS*

*FlashTek X-32VM*
*Protected Mode*

**Registers at call:**
AX = 3504h
BX = selector

**Return Registers:**
CF clear if successful
    ECX = absolute base address of selector
CF set on error (invalid selector)

**See also:** Function 3503h, DPMI INT 31h Function 0006h (Chapter 30)

### INT 21h Function 3505h
### *SET SELECTOR LIMIT*

*FlashTek X-32VM*
*Protected Mode*

**Registers at call:**
AX = 3505h
BX = selector
ECX = desired limit

**Return Registers:**
CF clear if successful
    ECX = actual limit set
CF set on error (no more selectors available)

**Details:** The limit will be rounded down to nearest 4K boundary if the requested limit is greater than 1MB.

**See also:** Function 3503h, DPMI INT 31h Function 0008h (Chapter 30)

### INT 21h Function 350Ah
### *PHYSICAL ADDRESS MAPPING*

*FlashTek X-32VM*
*Protected Mode*

**Registers at call:**
AX = 350Ah
EBX = absolute physical address
ECX = size in bytes of area to map

**Return Registers:**
CF clear if successful
CF set on error (insufficient memory or service refused by DPMI host)

**Details:** The application should not make repeated calls for the same physical address. There is no provision for unmapping memory.

### INT 21h Function 350Bh
### *UPDATE AND RETURN AVAILABLE FREE MEMORY*

*FlashTek X-32VM*
*Protected Mode*

**Registers at call:**
AX = 350Bh
DS = default selector for DS

**Return Registers:**
CF clear
EAX = maximum amount of memory which can be allocated via Function 350Ch

**See also:** Function 350Ch

### INT 21h Function 350Ch
### *ALLOCATE A BLOCK OF MEMORY*

*FlashTek X-32VM*
*Protected Mode*

**Registers at call:**
AX = 350Ch
ECX = size of block in bytes
DS = default DS

**Return Registers:**
CF clear if successful
    EAX = near pointer to new block
    EDX = new lowest legal value for stack
CF set on error (requested size not multiple of 4K)

**See also:** Functions 350Bh and 350Dh

### INT 21h Function 350Dh
### *RESERVE BLOCK OF MEMORY FOR 32-BIT STACK*

*FlashTek X-32VM*
*Protected Mode*

**Registers at call:**
AX = 350Dh
EBX = current ESP value
ECX = size of block in bytes
DS = default DS

**Return Registers:**
CF clear if successful
    EBX = new value for ESP
    EDX = suggested new limit for SS
CF set on error

**Details:** This function should only be called once during initialization.

**See also:** Functions 350Bh and 350Ch

## CHAPTER 34

# *OS/286 and OS/386*

OS/286 and OS/386 are 80286- and 80386-specific DOS extenders, respectively. Originally by AI Architects, they are now being marketed and further developed by Ergo Computing.

### INT 21h Function 4403h
### *OS/x86 - API*

*AI Architects*
*(undocumented)*

**Registers at call:**
AX = 4403h
BX = handle for device "AIA_OS"
CX = number of bytes to write (ignored)
DS:DX -> 12-byte buffer, first byte is command:
- 81h installation check
- 82h get API entry point
- 84h uninstall

**Return Registers:**
CF clear if successful
    AX = number of bytes actually written
CF set on error
    AX = error code (01h,05h,06h,0Ch,0Dh) (see Table 24-2)

**Details:** These functions are only available if the DOS extender was loaded as a device driver in CONFIG.SYS.

Called by TKERNEL (a licensed version of AI Architects/Ergo OS/x86).

**See also:** TKERNEL INT 2Fh Function FBA1h Subfunctions 0081h and 0082h (Chapter 36)

**34-1 Format of buffer on return:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 4 BYTEs | signature "IABH" |
| **---if function 81h---** | | |
| (no additional fields) | | |
| **---if function 82h---** | | |
| 04h | DWORD | pointer to API entry point (see Tables 36-8 through 36-47) |
| **---if function 84h---** | | |
| 04h | WORD | success indicator |
| 06h | WORD | segment of *unknown item* |
| 08h | WORD | segment of *unknown* memory block to free if nonzero |
| 0Ah | WORD | segment of *unknown* memory block to free if nonzero |

**34-2 OS/286 and OS/386 INT 21h Function Overview:**

| *Function* | *Description* |
|---|---|
| E0h | Initialize real procedure |
| E1h | Issue real procedure call |
| E2h | Set real procedure signal handler |

| | |
|---|---|
| E3h | Issue real interrupt (details below) |
| E400h | Chain to real-mode handler |
| E402h | Set protected-mode task gate |
| E403h | Remove protected-mode task gate |
| E500h | Heap management strategy |
| E501h | Force heap compaction |
| E6h | Issue real procedure signal from protected mode |
| E7h | Create code segment |
| E8h | Segment creation (details below) |
| E9h | Change segments (details below) |
| EAh | Allocate huge segment |
| EBxxh | OS/386 VMM (see Table 34-3) |
| ECh | Block transfer |
| EDh | Get segment or window descriptor |

**Note:** These functions may only be available in protected mode.
**Conflicts:** Digital Research DOS Plus (Chapter 26), DoubleDOS (Chapter 49), Viruses (Chapter 59), Novell NetWare (NI)

**34-3 OS/386 VMM INT 21h Function Overview:**

| *Function* | *Description* |
|---|---|
| EB00h | Get a page table entry by linear address |
| EB02h | Get a page table entry by 16-bit segment:offset |
| EB03h | Free mapped pages |
| EB04h | Get a page table entry by 32-bit segment:offset |
| EB05h | Map pages |
| EB06h | Lock pages in memory |
| EB07h | Unlock memory pages |

**Note:** These functions may only be available in protected mode.
**Conflicts:** DoubleDOS (Chapter 49), Novell NetWare (NI)

### INT 21h Function E3h
### *ISSUE REAL INTERRUPT*

*OS/286, OS/386*

**Registers at call:**
AH = E3h
AL = interrupt number
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** *This function may only be available in protected mode.*
**See also:** Function E1h, DPMI INT 31h Function 0300h (Chapter 30)

### INT 21h Function E8h
### *SEGMENT CREATION*

*OS/286, OS/386*

**Registers at call:**
AH = E8h
AL = type
- 00h data segment
- 01h data window/alias
- 02h real segment
- 03h real window/alias
- 06h shareable segment

*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Functions E7h and E9h

### INT 21h Function E9h
### *CHANGE SEGMENTS*

*OS/286, OS/386*
*Protected Mode*

**Registers at call:**
AH = E9h
AL = function
    01h change code segment parameters
    02h change data segment parameters
    05h adjust segment limit
    06h change segment base address
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Functions E7h, E8h, EAh, and EDh; DPMI INT 31h Functions 0007h and 0008h (Chapter 30)

### INT 2Fh Function A1h, Subfunction 0081h
### *INSTALLATION CHECK*

*Ergo DOS extenders*

**Registers at call:**
AH = A1h
BX = 0081h
AL = which
    FEh OS/286,OS/386
    FFh HummingBoard DOS extender
ES:DI -> 16-byte buffer

**Return Registers:**
if installed, first four bytes of ES:DI buffer are "IABH"

**Details:** Since Borland's TKERNEL is a licensed version, it is likely that Subfunctions 0082h and 0084h are present and function identically to the Function FBA1h Subfunction 008xh calls.

**See also:** Phar Lap Function ED00h (Chapter 33), TKERNEL Function FBA1h Subfunction 0081h (Chapter 36), DOS/16M INT 15h Function BF02h (Chapter 35)

## CHAPTER 35

# *Rational Systems DOS Extenders*

Rational Systems' DOS/16M is one of the oldest DOS extenders available, and is used in numerous commercial programs such as Lotus 1-2-3 version 3.0. A newer, 80386-specific DOS extender called DOS/4G has been available for some time, and comes in at least two flavors: DOS/4GW for the Watcom C compiler and DOS/4GX for use with Quarterdeck's DESQview/X (see Chapter 44).

---

### INT 15h Functions BF00h and BF01h *DOS/16M*
***UNDETERMINED FUNCTIONS***

| Registers at call: | Return Registers: |
|---|---|
| AX = BF00h and BF01h | *unknown* |
| *others, if any, unknown* | |

**Details:** Under DESQview/X 1.02 DVDOS4GX.DVR, these calls are identical to Function BF02h. Function BF01h is called by DOS/4GW.

**See also:** Function BF02h

---

### INT 15h Function BF02h *DOS/16M*
***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = BF02h | DX = nonzero if installed |
| DX = 0000h | DX:SI -> XBRK structure (see Table 35-1) |

**Details:** This function is also supported by DOS/4G.

**See also:** Functions BF01h and BFDCh, INT 21h Function FFh Subfunction 0Eh, DVDOS4GX Function BFDEh Subfunction 0000h (Chapter 44), Ergo INT 2Fh Function A1h (Chapter 34), DOS extender INT 2Fh Function F100h (Chapter 37), TKERNEL INT 2Fh Function FBA1h (Chapter 36)

**35-1 Format of XBRK structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | linear address of first available byte |
| 04h | DWORD | *linear address of last available byte + 1* |
| 08h | DWORD | *real-mode address of XBRK structure* |
| 0Ch | DWORD | *unknown* |
| 10h | 2 BYTEs | *unknown* |
| 12h | WORD | segment of *unknown item* |
| 14h | 8 BYTEs | *unknown* |
| 1Ch | 512 BYTEs | protected-mode IDT |
| 21Ch | N BYTEs | protected-mode GDT |

### INT 15h Function BF03h

*Rational Systems DOS/4GW*

***UNINSTALL***

**Registers at call:**
AX = BF03h
BX = PSP segment of extender
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** If BX is not the PSP segment of the extender, it passes the call down the INT 15h chain; this allows nested instances of the extender.

**See also:** Function BF06h

### INT 15h Function BF04h

*Rational Systems DOS/4GW*

***UNDETERMINED FUNCTION***

**Registers at call:**
AX = BF04h
BX = PSP segment of extender

**Return Registers:**
*none?*

**Details:** If BX is not the PSP segment of the extender, it passes the call down the INT 15h chain; this allows nested instances of the extender.

This function grabs INT 2Fh and installs handlers for INT 2Fh Function 1605h-1607h.

**See also:** INT 2Fh Function 1607h Subfunction 22C0h

### INT 15h Function BF05h

*Rational Systems DOS/4GW*

***INITIALIZE PROTECTED-MODE INTERFACE***

**Registers at call:**
AX = BF05h
BX = PSP segment of extender

**Return Registers:**
*none?*

**Details:** If BX is not the PSP segment of the extender, it passes the call down the INT 15h chain; this allows nested instances of the extender.

This function calls INT 67h Function DE01h under unknown conditions.

### INT 15h Function BF06h

*Rational Systems DOS/4GW*

***UNDETERMINED FUNCTION***

**Registers at call:**
AX = BF06h
BX = PSP segment of extender
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** If BX is not the PSP segment of the extender, it passes the call down the INT 15h chain; this allows nested instances of the extender.

**See also:** Function BF03h

### INT 15h Function BFDCh

*Rational Systems DOS/4GW*

***INSTALLATION CHECK***

**Registers at call:**
AX = BFDCh
DX = 0000h
SI = 0000h

**Return Registers:**
DX = nonzero if installed
    DX:SI -> XBRK structure (see Table 35-1)

**See also:** Function BF02h

### INT 21h Function FFh

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

***API***

**Registers at call:**
AH = FFh
DH = function (00h-17h) (see also separate entries below)
DL = subfunction or argument

**Return Registers:**
CF clear if valid function number
    AX = *status*
CF set if invalid function

**See also:** INT 15h Function BFDCh

---

### INT 21h Function FFh, Subfunction 02h
**SET UNKNOWN VALUE**

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh
DH = 02h
DL = *unknown*

**Return Registers:**
CF clear

---

### INT 21h Function FFh, Subfns 05h to 07h
**UNDETERMINED FUNCTIONS**

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh
DH = 05h to 07h
BX = *unknown*

**Return Registers:**
*unknown*

---

### INT 21h Function FFh, Subfunction 08h
**UNDETERMINED FUNCTION**

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh
DH = 08h
BX = *unknown*
CX = *unknown*
ES = *unknown*

**Return Registers:**
*unknown*

---

### INT 21h Function FFh, Subfunction 09h
**GET UNKNOWN VALUE**

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh
DH = 09h

**Return Registers:**
ES:BX -> *unknown*

---

### INT 21h Function FFh, Subfns 0Ah and 0Bh
**UNDETERMINED FUNCTIONS**

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh
DH = 0Ah and 0Bh
AL = *unknown*
BX = *unknown*
CX = *unknown*

**Return Registers:**
ES = *unknown* or 0000h

---

### INT 21h Function FFh, Subfunction 0Ch
**GET/SET UNKNOWN VALUE**

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh
DH = 0Ch
DL = *unknown* (00h or 01h)

**Return Registers:**
CF clear if successful
    AL = previous value of *unknown*
CF set on error (DL out of range)
    AX = FFFFh

---

### INT 21h Function FFh, Subfunction 0Dh
**UNDETERMINED FUNCTION**

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh
DH = 0Dh
*others, if any, unknown*

**Return Registers:**
*unknown*

---

**INT 21h Function FFh, Subfunction 0Eh**
***UNDETERMINED FUNCTION***

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh
DH = 0Eh

**Return Registers:**
DX:AX -> XBRK structure (see Table 35-1)
BX = *unknown*
CX = *unknown*

**See also:** INT 15h Function BF02h

---

**INT 21h Function FFh, Subfunction 0Fh**
***UNDETERMINED FUNCTION***

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh
DH = 0Fh
*others, if any, unknown*

**Return Registers:**
*unknown*

---

**INT 21h Function FFh, Subfunction 10h**
***UNDETERMINED FUNCTION***

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh
DH = 10h
AL = *unknown*
BX = *unknown*
CX = *unknown*
DI = *unknown*
SI = *unknown*

**Return Registers:**
*unknown*

**Details:** Among other things, this function frees two memory blocks via INT 21h Function 49h.

---

**INT 21h Function FFh, Subfunction 11h**
***NOP***

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh
DH = 11h

**Return Registers:**
none

---

**INT 21h Function FFh, Subfunction 12h**
***EXCHANGE UNKNOWN POINTERS***

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh
DH = 12h
DS:SI -> new *unknown*
ES:DI -> new *unknown*

**Return Registers:**
DS:SI -> previous *unknown*
ES:DI -> previous *unknown*

---

**INT 21h Function FFh, Subfunction 13h**
***UNDETERMINED FUNCTION***

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh
DH = 13h
AL = *unknown*
ES = *unknown*

**Return Registers:**
*unknown*

---

**INT 21h Function FFh, Subfunction 14h**
***UNDETERMINED FUNCTION***

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

**Registers at call:**
AH = FFh

| | **Return Registers:** |
|---|---|
| DH = 14h | CF clear |
| BX = *unknown* | AX = *unknown* |
| CX = *unknown* | DX = *unknown* |

---

### INT 21h Function FFh, Subfunction 15h
### *GET UNKNOWN FUNCTIONS*

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = FFh | CF clear |
| DH = 15h | DX:AX -> FAR function for *unknown* |
| | CX:BX -> FAR function for *unknown* |
| | SI:DI -> FAR function for *unknown* |

---

### INT 21h Function FFh, Subfunction 16h
### *GET UNKNOWN VALUE*

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = FFh | AX = *unknown* |
| DH = 16h | |

---

### INT 21h Function FFh, Subfunction 17h
### *UNDETERMINED FUNCTION*

*Rational Systems DOS/4GW*
*Protected Mode (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = FFh | *unknown* |
| DH = 17h | |
| AL = *unknown* | |
| DL = *unknown* | |

---

### INT 21h Function FF00h, Subfunction 0078h
### *INSTALLATION CHECK*

*Rational Systems DOS/4G*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = FF00h | AL nonzero if installed |
| DX = 0078h | GS = segment of kernel if nonzero |

**See also:** INT 15h Function BF02h

---

### INT 2Fh Function 1607h, Subfunction 22C0h
### *UNDETERMINED FUNCTION*

*Rational Systems DOS/4GW*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1607h | *unknown* |
| BX = 22C0h | |
| *others, if any, unknown* | |

**See also:** INT 15h Functions BF02h and BF04h

## CHAPTER 36

# *Borland DOS Extenders*

With the introduction of Turbo Pascal version 6 and Borland C++ in February 1991, Borland International added a DOS extender to permit its compilers to compile files too large to be compiled in real mode. The protected-mode versions of the compilers will automatically load the extender, but the DOS extender may also be installed as a TSR to speed up the loading of the compiler.

## DPMILOAD

DPMILOAD is the DOS extender provided with the most recent versions of Borland's C++ compiler. The version of DPMILOAD distributed with BC++ v2.0 identifies itself as version 1.000, while the version distributed with BC++ 3.0 identifies itself as version 1.0; the former is 10864 bytes, the latter 22180 bytes. The BC2.0 version is only a DPMI loader (requiring a separate host), while the BC3.0 version also adds a DPMI host and DOS extender.

---

### INT 2Fh Function FB42h, Subfunction 0001h
### *INSTALLATION CHECK*

*Borland C++ DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0001h

**Return Registers:**
AX = *version number* (AL=major, AH=minor)
CX = next-selector increment
---**BC2.0**---
ES:BX -> 80-byte buffer for *unknown*
DX = DPMI version
---**BC3.0**---
BX = *unknown* (0000h)
DX = *unknown*
ES:SI -> list of valid selectors *unknown*

**Details:** The BC++ 2.0 version displays an error message if called with BX values other than 0001h-0008h.

**See also:** Function FB42h Subfunctions 0014h and 1001h, Function FB43h, DPMI Function 1687h (Chapter 30)

---

### INT 2Fh Function FB42h, Subfunction 0002h
### *ALLOCATE MEMORY*

*Borland C++ 2.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0002h
CX = size in bytes
DX = bit flags

bit 2: set to allocate DOS memory, clear for DPMI memory
SI = selector of descriptor to be modified to access allocated memory
DI = selector of a second descriptor to be modified

**Return Registers:**
AX = *unknown* or 0000h on error
CX:DX = linear base address of DPMI memory block
SI:DI = handle for DPMI memory block or FFFFh:FFFFh
*others, if any, unknown*

**Details:** Two segment descriptors may be set if a code and an aliased data segment are required; if only one descriptor is needed, SI should equal DI on entry

**BUG:** When allocating DOS memory, the code computes the linear address by multiplying the segment number by 4 rather than shifting by 4.

**See also:** Function FB42h Subfunctions 0003h and 0008h, DPMI INT 31h Function 0501h (Chapter 30)

## INT 2Fh Function FB42h, Subfunction 0002h
## *ALLOCATE MEMORY*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0002h
ES:SI -> memory block info (see Table 36-1)

**Return Registers:**
*unknown*

**See also:** Function FB42h Subfunctions 0003h and 0008h, DPMI INT 31h Function 0501h (Chapter 30)

### 36-1 Format of memory block info:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | flags |
| | | bit 0: set if data segment rather than code segment |
| | | bit 1: information valid |
| | | bit 2: set if DOS memory block rather than DPMI memory block |
| | | bit 4: *unknown* |
| | | bit 15: set if *no LDT selectors for memory block* |
| 02h | DWORD | block size in bytes |
| ---DPMI memory block--- | | |
| 06h | DWORD | DPMI memory block handle |
| ---DOS memory block--- | | |
| 06h | WORD | real-mode segment of memory block |
| 08h | WORD | selector of memory block |
| --- | | |
| 0Ah | DWORD | linear address of memory |
| 0Eh | WORD | memory operation error code |
| | | 0008h no more free LDT descriptors |
| ---if flags bit 0 clear--- | | |
| 10h | WORD | code segment selector for memory block or 0000h or FFFFh |
| 12h | WORD | data alias selector for memory block or 0000h or FFFFh |
| ---if flags bit 0 set--- | | |
| 10h | WORD | data segment selector for memory block or 0000h or FFFFh |
| 12h | WORD | *unused* |

## INT 2Fh Function FB42h, Subfunction 0003h
## *GET AVAILABLE MEMORY*

*Borland C++ DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0003h

**Return Registers:**
DX:AX = size of largest free block in paragraphs
0000h:0000h on error (BC3.0 version only)

**Details:** AX and DX are destroyed on error, but no other error indicator is returned, under the BC++ 2.0 version of DPMILOAD.
**See also:** Function FB42h Subfunction 0002h

---

### INT 2Fh Function FB42h, Subfunction 0004h
### *LOAD PROTECTED-MODE EXECUTABLE*

*Borland C++ DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0004h
DS:DX -> ASCIZ filename of protected-mode executable

**Return Registers:**
CX = selector of *unknown* or 0000h
---**BC3.0**---
DX = status (0000h,FFF4h,*others*) (see Table 36-2)

**Details:** The filename may also be terminated by a CR rather than a NUL under the BC++ 3.0 version of DPMILOAD.

**36-2 Values for status:**

| | |
|---|---|
| 0000h | successful |
| 0001h | *unknown* failure |
| 0002h | invalid selector |
| 0004h | unknown error |
| 0008h | *no more LDT descriptors available* |
| FFDEh | unable to set descriptor |
| FFDFh | unable to get segment base address |
| FFE0h | *unknown* |
| FFF2h | invalid parameter value |
| FFF4h | component of filename too long (name not in 8.3 format) |
| FFF5h | pathname too long (>79 chars) |
| FFF6h | *unknown* |
| FFF8h | *unknown* |
| FFF9h | index out of range |
| FFFAh | *unknown* |
| FFFCh | *invalid access to code segment* |
| FFFEh | *unknown* |
| FFFFh | general error |

---

### INT 2Fh Function FB42h, Subfunction 0005h
### *GET ADDRESS OF UNKNOWN ITEM BY NAME*

*Borland C++ DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0005h
CX = selector of DPMILOAD data (see Table 36-3)
DS:DX -> ASCIZ or CR-terminated name of *unknown* (case ignored)

**Return Registers:**
DX = status (see Table 36-2)
  0000h successful
    AX:BX -> *unknown* FAR function (called with two words on top of stack)
  else
    BX destroyed

**See also:** Function FB42h Subfunctions 0006h and 000Eh

**36-3 Format of DPMILOAD data:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | *unknown* |
| 0Ch | WORD | *unknown* bit flags |
| 0Eh | 14 BYTEs | *unknown* |
| 1Ch | WORD | number of memory control records |
| 1Eh | 25 BYTEs | *unknown* |

| | | |
|---|---|---|
| 37h | BYTE | *unknown* bit flags<br>bit 4: *data valid* |
| 38h | 4 BYTEs | *unknown* |
| 3Ch | WORD | *unknown* |
| 3Eh | 12 BYTEs | *unknown* |
| 46h | BYTE | *unknown* counter |
| 47h | BYTE | *unknown* |
| 48h | BYTE | *unknown* |
| 49h | BYTE | *unknown* |
| 4Ah | WORD | *unknown* |
| 4Ch | 2 BYTEs | *unknown* |
| 4Eh | WORD | offset of array of 64-byte memory control records |
| 52h | WORD | offset of name list (see Table 36-4) |
| 54h | 4 BYTEs | *unknown* |
| 58h | WORD | offset of array of 6-byte objects |
| 5Ah | 8 BYTEs | *unknown* |
| 62h | 9 BYTEs | ASCIZ name for *unknown item* |
| 6Bh | 9 BYTEs | ASCIZ name for *unknown item* |
| | *others, if any, unknown* | |

**36-4 Format of name list entry [array]:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | length of name (00h if end of array) |
| 01h | N BYTEs | name |
| N+1 | WORD | 1-based index into array of unknown 6-byte objects |

**36-5 Format of 6-byte objects:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | *unknown* |
| 01h | BYTE | *unknown* |
| 02h | BYTE | *unknown* |
| 03h | BYTE | 1-based index of memory control record |
| 04h | WORD | *unknown* |

**36-6 Format of memory control record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 20 BYTEs | memory block info (see Table 36-1) |
| 14h | 6 BYTEs | *unknown* |
| 1Ah | BYTE | *unknown* |
| 1Bh | 2 BYTEs | *unknown* |
| 1Dh | BYTE | *unknown* bit flags |
| 1Eh | 14 BYTEs | *unknown* |
| 2Ch | DWORD | pointer to *unknown* memory control record or 0000h:0000h |
| 30h | DWORD | pointer to *unknown* memory control record or 0000h:0000h |
| 34h | DWORD | pointer to *next* memory control record or 0000h:0000h |
| 38h | DWORD | pointer to *previous* memory control record or 0000h:0000h |
| 3Ch | 4 BYTEs | *unknown* |

**Note:** the pointers at offsets 2Ch and 30h form a doubly-linked list, as do the pointers at offsets 34h and 38h

## INT 2Fh Function FB42h, Subfunction 0005h

### INT 2Fh Function FB42h, Subfunction 0006h
### *GET ADDRESS OF UNKNOWN ITEM BY NUMBER*

*Borland C++ DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0006h
CX = selector of DPMILOAD data
  (see Table 36-3)
DX = 1-based index into array of
  *unknown* 6-byte objects

**Return Registers:**
DX = status (see Table 36-2)
  0000h successful
    AX:BX -> *unknown* FAR function (called
      with two words on top of stk)
  else
    BX destroyed

**See also:** Function FB42h Subfunctions 0005h and 000Eh

### INT 2Fh Function FB42h, Subfunction 0007h
### *UNDETERMINED FUNCTION*

*Borland C++ 2.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0007h
CX = selector of *unknown*

**Return Registers:**
*unknown*

### INT 2Fh Function FB42h, Subfunction 0007h
### *UNDETERMINED FUNCTION*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0007h
CX = selector of DPMILOAD data
  (see Table 36-3)
*others, if any, unknown*

**Return Registers:**
DX = status (see Table 36-2)
  0000h successful
    AX = *unknown*

### INT 2Fh Function FB42h, Subfunction 0008h
### *FREE MEMORY BLOCK*

*Borland C++ 2.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0008h
CX = bit flags
  bit 2: set if DPMI memory, clear if DOS memory
DX = selector of DOS memory block
SI:DI = handle of DPMI memory block

**Return Registers:**
DX = 0000h on error, unchanged if succcessful

**See also:** Function FB42h Subfunction 0002h

### INT 2Fh Function FB42h, Subfunction 0008h
### *FREE MEMORY BLOCK*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0008h
ES:SI -> memory block info (see Table 36-1)

**Return Registers:**
*unknown*

**See also:** Function FB42h Subfunction 0009h

### INT 2Fh Function FB42h, Subfunction 0009h
### *RESIZE MEMORY BLOCK*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0009h
ES:SI -> memory block info (see

Table 36-1)
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** BC++ 3.0 Function FB42h Subfunction 0008h

---

### INT 2Fh Function FB42h, Subfunction 000Ah
### *INITIALIZE DPMI HOST AND SPAWN SUBSHELL*

*Borland C++ 3.0 DPMILOAD.EXE*
*Real Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 000Ah
CX = 0001h
DX = *unknown*
SI = *unknown*

**Return Registers:**
after user exits subshell

**Details:** This call is used by DPMIRES; unlike most of the DPMILOAD calls, this function is not available in protected mode.

The BC2.0 version (which does not support this call) of DPMILOAD is purely a DPMI loader, while the BC3.0 version also adds a DPMI host and DOS extender.

**Conflicts:** AutoBraille (Chapter 23), Jot-It! (Chapter 66)
**See also:** Function FB42h Subfunctions 0004h and 0015h

---

### INT 2Fh Function FB42h, Subfunction 000Bh
### *UNUSED*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 000Bh

**Return Registers:**
none

---

### INT 2Fh Function FB42h, Subfunction 000Ch
### *FREE DESCRIPTORS FOR MEMORY BLOCK*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 000Ch *unknown*
ES:SI -> *memory block info* (see Table 36-1)

**Return Registers:**
DX = *status*

**See also:** Function FB42h Subfunction 000Fh

---

### INT 2Fh Function FB42h, Subfunction 000Dh
### *SIMULATE REAL MODE INTERRUPT*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 000Dh
CX = number of words to copy from protected-mode to real mode stack
DL = interrupt number
DH = flags
  bit 0: reset the interrupt controller and A20 line
ES:DI -> real-mode call structure (see Table 30-8)

**Return Registers:**
CX = status
  0000h successful
  0001h failed

**See also:** DPMI INT 31h Function 0300h (Chapter 30)

---

### INT 2Fh Function FB42h, Subfunction 000Eh
### *GET ADDRESS OF UNKNOWN ITEM*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 000Eh
DS:DX -> ASCIZ or CR-terminated name of *unknown*

**Return Registers:**
CX = selector of DPMILOAD data (see Table 36-3) corresponding to name, 0000h on error

**See also:** Function FB42h Subfunctions 0006h and 001Fh

---

**INT 2Fh Function FB42h, Subfunction 000Fh** *Borland C++ 3.0 DPMILOAD.EXE*
***REALLOCATE LDT DESCRIPTORS TO MEMORY BLOCK*** *Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 000Fh
ES:SI -> memory block info (see Table 36-1)

**Return Registers:**
*unknown*

**See also:** Function FB42h Subfunction 000Ch

---

**INT 2Fh Function FB42h, Subfunction 0010h** *Borland C++ 3.0 DPMILOAD.EXE*
***CONVERT SEGMENT TO SELECTOR*** *Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0010h
DX = segment number

**Return Registers:**
CX = status (see also Table 36-2)
  0000h successful
    DX = selector number for descriptor
  0008h failed

**See also:** Function FB42h Subfunction 0023h

---

**INT 2Fh Function FB42h, Subfunction 0011h** *Borland C++ 3.0 DPMILOAD.EXE*
***UNDETERMINED FUNCTION*** *Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0011h
CX = selector of DPMILOAD data
  (see Table 36-3)
*others, if any, unknown*

**Return Registers:**
DX = status (see also Table 36-2)
  0000h successful
    AX:BX -> *unknown* name
  0002h invalid selector
  FFFEh *unknown* error

---

**INT 2Fh Function FB42h, Subfunction 0012h** *Borland C++ 3.0 DPMILOAD.EXE*
***UNDETERMINED FUNCTION*** *Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0012h
CX = selector for *unknown*

**Return Registers:**
CX = selector for *unknown*

---

**INT 2Fh Function FB42h, Subfunction 0013h** *Borland C++ 3.0 DPMILOAD.EXE*
***UNDETERMINED FUNCTION*** *Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0013h
CX = selector of DPMILOAD data
  (see Table 36-3)
DX = 1-based index of *unknown*

**Return Registers:**
CX = status (see also Table 36-2)
  0000h successful
    BX = *unknown* or 0000h
  0002h invalid selector
  FFF9h *unknown* error

---

**INT 2Fh Function FB42h, Subfunction 0014h** *Borland C++ 3.0 DPMILOAD.EXE*
***INSTALLATION CHECK*** *Real Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0014h
CX = 0001h

**Return Registers:**
BX = 0000h if installed

**Details:** Unlike most of the DPMILOAD functions, this call is available only in real or V86 mode.
**Conflicts:** AutoBraille (Chapter 23), Jot-It! (Chapter 66)
**See also:** Function FB42h Subfunctions 0001h and 000Ah

---

**INT 2Fh Function FB42h, Subfunction 0015h**
***UNINSTALL***

*Borland C++ 3.0 DPMILOAD.EXE*
*Real Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0015h
CX = 0001h

**Return Registers:**
*unknown*

**Details:** Unlike most of the DPMILOAD functions, this call is available only in real or V86 mode.
**Conflicts:** AutoBraille (Chapter 23), Jot-It! (Chapter 66)
**See also:** Function FB42h Subfunction 000Ah

---

**INT 2Fh Function FB42h, Subfunction 0016h**
***GET UNKNOWN ITEM***

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0016h
CX = selector of DPMILOAD data
(see Table 36-3)

**Return Registers:**
DX = status (see also Table 36-2)
0000h successful
CX = *unknown*

---

**INT 2Fh Function FB42h, Subfunction 0017h**
***UNDETERMINED FUNCTION***

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0017h
CX = *unknown*
DX = *unknown*
*others, if any, unknown*

**Return Registers:**
DX = status (0000h,0001h) (see Table 36-2)

---

**INT 2Fh Function FB42h, Subfunction 0018h**
***SET UNKNOWN ITEM***

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0018h
CX = *unknown*

**Return Registers:**
none

---

**INT 2Fh Function FB42h, Subfunction 0019h**
***UNDETERMINED FUNCTION***

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0019h
CX = selector for *unknown*
*others, if any, unknown*

**Return Registers:**
DX = status (see also Table 36-2)
0000h successful
CX = selector for *unknown*

---

**INT 2Fh Function FB42h, Subfunction 001Ah**
***UNDETERMINED FUNCTION***

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 001Ah
CX = selector for *unknown*
*others, if any, unknown*

**Return Registers:**
DX = status (see also Table 36-2)
0000h successful
0004h failed
CX:BX -> *unknown*

### INT 2Fh Function FB42h, Subfunction 001Bh
### *UNDETERMINED FUNCTION*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 001Bh
CX = selector of DPMILOAD data (see Table 36-3)
DX = offset of *unknown*

**Return Registers:**
DX = status (see also Table 36-2)
  0000h successful
    BX = selector for *unknown*
    CX = selector for *unknown*
  0002h invalid selector

### INT 2Fh Function FB42h, Subfunction 001Ch
### *UNDETERMINED FUNCTION*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 001Ch
ES = selector for DPMILOAD data (see Table 36-3)
CX = 1-based index of *unknown*
DX = 1-based index of *unknown*

**Return Registers:**
DX = status (0000h,0002h,FFF9h) (see Table 36-2)

### INT 2Fh Function FB42h, Subfunction 001Dh
### *GET UNKNOWN VALUE*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 001Dh

**Return Registers:**
CX:DX = *unknown*

### INT 2Fh Function FB42h, Subfunction 001Eh
### *UNDETERMINED FUNCTION*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 001Eh
CX = *unknown*
*others, if any, unknown*

**Return Registers:**
DX = status (see also Table 36-2)
  0000h successful
  FFF7h *unknown* error
CX:BX -> *unknown*

### INT 2Fh Function FB42h, Subfunction 001Fh
### *GET ADDRESS OF UNKNOWN ITEM*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 001Fh
DS:DX -> 8-character name of *unknown*
*others, if any, unknown*

**Return Registers:**
CX = selector of DPMILOAD data (see Table 36-3) for *unknown*; 0000h on error

**See also:** Function FB42h Subfunction 000Eh

### INT 2Fh Function FB42h, Subfunction 0020h
### *NULL FUNCTION*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0020h

**Return Registers:**
DX = *unknown* (always 0000h)

### INT 2Fh Function FB42h, Subfunction 0021h
### *GET PROCESSOR EXCEPTION HANDLER VECTOR*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h

BX = 0021h
CL = exception number (00h-1Fh)

**Return Registers:**
DX = status (see also Table 36-2)
0000h successful
AX:BX = selector:offset of handler
FFF2h unable to get exception handler vector

**See also:** Function FB42h Subfunctions 0022h and 0024h, DPMI INT 31h Function 0202h (Chapter 30)

---

### INT 2Fh Function FB42h, Subfunction 0022h
### *SET PROCESSOR EXCEPTION HANDLER VECTOR*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0022h
CL = exception number (00h-1Fh)
SI:DX = selector:offset of new handler

**Return Registers:**
DX = status (0000h,0004h,FFF2h) (see Table 36-2)

**See also:** Function FB42h Subfunctions 0021h and 0025h, DPMI INT 31h Function 0203h (Chapter 30)

---

### INT 2Fh Function FB42h, Subfunction 0023h
### *CONVERT SELECTOR TO SEGMENT NUMBER*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0023h
CX = selector

**Return Registers:**
DX = status (see also Table 36-2)
0000h successful
CX = real-mode segment number
FFF2h descriptor has invalid base address for real-mode segment

**See also:** Function FB42h Subfunction 0010h

---

### INT 2Fh Function FB42h, Subfunction 0024h
### *GET PROTECTED-MODE INTERRUPT VECTOR*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0024h
CL = interrupt number

**Return Registers:**
DX = status (0000h) (see also Table 36-2)
AX:BX = selector:offset of handler

**See also:** Function FB42h Subfunction 0025h, DPMI INT 31h Function 0204h (Chapter 30)

---

### INT 2Fh Function FB42h, Subfunction 0025h
### *SET PROTECTED-MODE INTERRUPT VECTOR*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0025h
CL = interrupt number
SI:DX = selector:offset of new handler

**Return Registers:**
DX = status (0000h,0004h,FFF2h) (see Table 36-2)

**See also:** Function FB42h Subfunction 0024h, DPMI INT 31h Function 0205h (Chapter 30)

---

### INT 2Fh Function FB42h, Subfunction 0026h
### *UNDETERMINED FUNCTION*

*Borland C++ 3.0 DPMILOAD.EXE*
*Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0026h
CX = selector of DPMILOAD data (see Table 36-3)
DX = 1-based index of *unknown*
*others, if any, unknown*

**Return Registers:**
DX = status (0000h,0002h,FFF9h) (see Table 36-2)
0000h successful
BX = offset of *unknown* within data structure

---

**INT 2Fh Function FB42h, Subfunction 0027h** *Borland C++ 3.0 DPMILOAD.EXE*
***GET UNKNOWN ITEM*** *Protected Mode (undocumented)*

**Registers at call:**
AX = FB42h
BX = 0027h
CX = selector of DPMILOAD data
  (see Table 36-3)
DX = offset of *unknown*

**Return Registers:**
DX = status (see also Table 36-2)
  0000h successful
    BX = *unknown*

---

**INT 2Fh Function FB42h, Subfunction 0080h** *UNKNOWN*
***CALLED BY Borland C++ 3.0 DPMILOAD.EXE*** *(undocumented)*

**Registers at call:**
AX = FB42h
BX = 0080h
*others, if any, unknown*

**Return Registers:**
AX = *unknown*
*others, if any, unknown*

---

**INT 2Fh Function FB42h, Subfunction 0081h** *UNKNOWN*
***CALLED BY Borland C++ 3.0 DPMILOAD.EXE*** *(undocumented)*

**Registers at call:**
AX = FB42h
BX = 0081h
*others, if any, unknown*

**Return Registers:**
AX = *unknown*
*others, if any, unknown*

---

**INT 2Fh Function FB43h** *Borland C++ 3.0 DPMILOAD.EXE*
***NULL FUNCTION*** *Protected Mode (undocumented)*

**Registers at call:**
AX = FB43h
BX = subfunction (at least
  0000h-000Eh)

**Return Registers:**
*unknown*
(DPMILOAD itself returns nothing)

**Details:** This function is only present in protected mode; it does nothing but an immediate IRET.

DPMILOAD.EXE itself calls various subfunctions:
subfunction 0004h is called with CX=selector of *unknown item*, DI=selector of DPMILOAD data
subfunction 0008h is called with CX=selector of DPMILOAD data.

**See also:** Function FB42h Subfunction 0001h

## RTM

RTM is the Run Time Manager included with Borland Pascal 7.0. It performs much the same function as the Borland C++ 3.x DPMILOAD, providing a DOS extender and a DPMI host if required.

---

**INT 2Fh Function FB42h, Subfunction 1001h** *Borland RTM.EXE 1.0*
*INSTALLATION CHECK* *(undocumented)*

**Registers at call:**
AX = FB42h
BX = 1001h

**Return Registers:**
BX = 0000h

**See also:** Function FB42h Subfunctions 0001h, 1002h, and 1003h

---

**INT 2Fh Function FB42h, Subfunction 1002h** *Borland RTM.EXE 1.0*
***UNDETERMINED FUNCTION*** *(undocumented)*

**Registers at call:**
AX = FB42h

BX = 1002h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function FB42h Subfunction 1001h

---

### INT 2Fh Function FB42h, Subfunction 1003h
### *UNDETERMINED FUNCTION*

*Borland RTM.EXE 1.0*
*(undocumented)*

**Registers at call:**
AX = FB42h
BX = 1003h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function FB42h Subfunction 1001h

## TKERNEL

TKERNEL is the DOS extender provided with Borland's C++ version 2.0; it is a licensed version of AI Architects/Ergo's OS/x86. TKERNEL was present only in Borland C++ 2.0; with version 3.0, the DOS extender was moved into DPMILOAD.

---

### INT 2Fh Function FBA1h, Subfunction 0081h
### *INSTALLATION CHECK*

*TKERNEL*
*(undocumented)*

**Registers at call:**
AX = FBA1h
BX = 0081h
ES:DI -> 16-byte buffer

**Return Registers:**
if installed, first four bytes of ES:DI buffer are "IABH"

**See also:** Function FBA1h Subfunctions 0082h and 0084h, Ergo Function A1h (Chapter 34), DOS extender Function F100h (Chapter 37), DOS/16M INT 15h Function BF02h (Chapter 35)

---

### INT 2Fh Function FBA1h, Subfunction 0082h
### *GET ENTRY POINT*

*TKERNEL*
*(undocumented)*

**Registers at call:**
AX = FBA1h
BX = 0082h
ES:DI -> response buffer (see Table 36-7)

**Return Registers:**
ES:DI buffer filled

**Details:** BX may be destroyed by any of the API calls listed in Tables 36-8 through 36-47. API calls with AX set to any value other than those listed here return AX = 0001h.

**See also:** Function FBA1h Subfunctions 0081h and 0084h

**36-7 Format of response buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 4 BYTEs | signature "IABH" |
| 04h | DWORD | pointer to FAR extender entry point (see Tables 36-8 to 36-47) |

**36-8 TKERNEL Function 0000h calling convention:**

**Registers at call:**
AX = 0000h *initialize*
STACK: WORD *unknown*

**Return Registers:**
AX = *status*
STACK unchanged

**36-9 TKERNEL** *"get version"* **calling convention:**

**Registers at call:**
AX = 0001h

**Return Registers:**
AX = 0200h for v2.0.34

**36-10 TKERNEL Function 0002h calling convention:**

**Registers at call:**
AX = 0002h get *unknown*

**Return Registers:**
AX = *unknown* (011Eh or 0182h seen)

### 36-11 TKERNEL "load protected-mode executable" calling convention:

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 0003h | | | AX = *status* |
| STACK: | DWORD | -> ASCIZ filename of executable | STACK unchanged |
| | DWORD | *unknown* | |
| | DWORD | -> program arguments (counted string plus CR) | |
| | DWORD | -> environment for protected-mode executable (terminated with two consecutive NULs) | |
| | DWORD | -> WORD buffer for *unknown* | |

### 36-12 TKERNEL "get descriptor" calling convention:

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 0004h | | | STACK unchanged |
| STACK: | WORD | selector for which to get descriptor | CF clear if successful<br>buffer filled |
| | WORD | segment number (when running in real mode) | CF set on error<br>*AX destroyed* |
| | DWORD | -> buffer for descriptor | |

### 36-13 TKERNEL Function 0005h calling convention:

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 0005h *unknown* | | | AX = *status* |
| STACK: | WORD | selector for *unknown* | STACK unchanged |
| | WORD | *subfunction number*<br>0000h *run previously-loaded program*<br>0001h *unknown* (similar to 0000h)<br>0002h *unknown*<br>0003h *unknown*<br>0005h *unknown* (similar to 0000h and 0001h) | |

### 36-14 TKERNEL Function 0006h calling convention:

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 0006h *unknown* | | | AX = *unknown* |
| STACK: | WORD | *unknown* | STACK unchanged |
| | DWORD | -> WORD (call)<br>max iterations of *unknown*<br>(return) remaining iterations | |

### 36-15 TKERNEL Function 0007h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 0007h unused | AX = 0001h |

### 36-16 TKERNEL Function 0008h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 0008h unused | AX = 0001h |

**36-17 TKERNEL "copy protected-mode memory into conventional memory" calling convention:**

**Registers at call:**
AX = 0009h

| STACK: | | |
|---|---|---|
| | WORD | selector for source segment |
| | WORD | *segment of source if in real mode* |
| | DWORD | offset of source |
| | WORD | number of bytes to copy |
| | DWORD -> | low-memory destination |

**Return Registers:**
AX = status
STACK unchanged

**36-18 TKERNEL "copy conventional memory into protected-mode memory" calling convention:**

**Registers at call:**
AX = 000Ah

| STACK: | | |
|---|---|---|
| | WORD | selector for destination segment |
| | WORD | *segment of destination if in real mode* |
| | DWORD | offset of destination |
| | WORD | number of bytes to copy |
| | DWORD | -> low-memory source |

**Return Registers:**
AX = status
STACK unchanged

**36-19 TKERNEL Function 000Bh calling convention:**

**Registers at call:**
AX = 000Bh get *unknown* pointers

| STACK: | | |
|---|---|---|
| | WORD | desired pointer<br>0000h get *unknown*<br>0002h get protected-mode CR3<br>0003h get 4K page table buffer pointer<br>else Return: DX:AX = FFFFh:FFFFh |

**Return Registers:**
DX:AX = requested pointer
STACK unchanged

**36-20 TKERNEL Function 000Ch calling convention:**

**Registers at call:**
AX = 000Ch set *unknown* pointers

| STACK: | | |
|---|---|---|
| | WORD | desired pointer<br>0000h set *unknown*<br>0002h set protected-mode CR3<br>0003h set 4K page table buffer pointer<br>else ignore |
| | DWORD | new value for pointer |

**Return Registers:**
STACK unchanged

**36-21 TKERNEL Function 000Dh calling convention:**

**Registers at call:**
AX = 000Dh get *unknown* pointers

| STACK: | | |
|---|---|---|
| | WORD | desired pointer<br>0000h get *unknown*<br>0001h get *unknown*<br>0002h get *unknown*<br>0003h get *unknown* |

## INT 2Fh Function FBA1h, Subfunction 0082h

| | | | Return Registers: |
|---|---|---|---|
| | | 0004h get *unknown* | DX:AX = desired pointer |
| | | 0005h get *unknown* | STACK unchanged |
| | | 0006h get *unknown* | |
| | | 0007h get *unknown* | |
| | | else Return: DX:AX = FFFFh:FFFFh | |

**36-22 TKERNEL Function 000Eh calling convention:**

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 000Eh set *unknown* pointer | | | STACK unchanged |
| STACK: | WORD | desired pointer | |
| | | 0000h set *unknown* | |
| | | 0001h set *unknown* | |
| | | 0002h set *unknown* | |
| | | 0003h set *unknown* | |
| | | 0004h set *unknown* | |
| | | 0005h set *unknown* | |
| | | 0006h set *unknown* | |
| | | 0007h set *unknown* | |
| | | else Return: DX:AX = FFFFh:FFFFh | |

**36-23 TKERNEL Function 000Fh calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 000Fh get *unknown* | AX = *unknown* (seen 0008h) |

**36-24 TKERNEL Function 0010h calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 0010h get *unknown* | AX = *unknown* |

**36-25 TKERNEL "determine whether selector is valid" calling convention:**

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 0011h | | | AX = selector or 0000h if invalid |
| STACK: | WORD | possible selector | STACK unchanged |

**36-26 TKERNEL "get physical address" calling convention:**

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 0012h | | | DX:AX = 32-bit physical address or 00000000h on error |
| STACK: | WORD | selector for desired segment | BX destroyed |
| | WORD | segment number if in real mode | STACK unchanged |
| | DWORD | offset within segment | |

**36-27 TKERNEL Function 0013h calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 0013h *unknown* | (see also Table 36-26) |

**Details:** This function normally jumps to the code for Function 0012h.

**36-28 TKERNEL "copy protected-mode memory to conventional memory" calling convention:**

| Registers at call: | | |
|---|---|---|
| AX = 0014h copy protected-mode memory to conventional memory, with *unknown* | | |
| STACK: | WORD | selector for source segment |

WORD *segment of source if in real mode*
DWORD offset of source
WORD number of bytes to copy
DWORD -> low-memory destination.

**Return Registers:**
AX = *status*
STACK unchanged

### 36-29 TKERNEL "copy conventional memory to protected-mode memory" calling convention:

**Registers at call:**
AX = 0015h copy conventional memory to protected-mode memory, with *unknown*
STACK: WORD selector for destination segment
WORD *segment of destination if in real mode*
DWORD offset of destination
WORD number of bytes to copy
DWORD -> low-memory source

**Return Registers:**
AX = *status*
STACK unchanged

### 36-30 TKERNEL Function 0016h calling convention:

**Registers at call:**
AX = 0016h set *unknown* pointer
STACK: WORD unused
DWORD -> *unknown* or 0000h:0000h

**Return Registers:**
AX = 0000h
STACK unchanged

### 36-31 TKERNEL "*allocate real-mode procedure*" calling convention:

**Registers at call:**
AX = 0017h
STACK: DWORD ASCIZ name of procedure
DWORD *unknown*
DWORD address of subroutine to invoke

**Return Registers:**
AX = status
0032h procedure by that name exists
0033h no more real-mode procedures available
DX destroyed
STACK unchanged

### 36-32 TKERNEL Function 0018h calling convention:

**Registers at call:**
AX = 0018h unused

**Return Registers:**
AX = 0001h

### 36-33 TKERNEL "get parameter block" calling convention:

**Registers at call:**
AX = 0019h

**Return Registers:**
DX:AX -> parameter block (format unknown at this time, but 92h bytes)
(preceded by signature "!!PARAM-BLOCK!!")

### 36-34 TKERNEL Function 001Ah calling convention:

**Registers at call:**
AX = 001Ah get *unknown*

**Return Registers:**
AX = *unknown* (0148h seen)

### 36-35 TKERNEL "*free real-mode procedure*" calling convention:

**Registers at call:**
AX = 001Bh
STACK: DWORD -> ASCIZ name of procedure

**Return Registers:**
*unknown*
STACK unchanged

## INT 2Fh Function FBA1h, Subfunction 0082h

**36-36 TKERNEL "check whether packets from protected mode task pending" calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 001Ch | AX = 0001h if packets pending, 0000h if not |

**36-37 TKERNEL Function 001Dh calling convention:**

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 001Dh set *unknown* | | | AX,BX destroyed |
| STACK: | DWORD | *unknown* or 0000h:0000h | STACK unchanged |

**36-38 TKERNEL Function 001Eh calling convention:**

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 001Eh *unknown* | | | AX,BX,CX,DX destroyed |
| STACK: | WORD | *unknown* (high byte Ignored) | data structure updated |
| | DWORD | -> data structure (see Table 36-48) | STACK unchanged |

**36-39 TKERNEL Function 001Fh calling convention:**

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 001Fh set *unknown* | | | AX destroyed |
| STACK: | WORD | *unknown* (set to 0001h if zero) | STACK unchanged |

**36-40 TKERNEL Function 0020h calling convention:**

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 0020h *unknown* | | | AX = *unknown* |
| STACK: | DWORD | -> *unknown* (8 bytes of data) | STACK unchanged |

**36-41 TKERNEL Function 0021h calling convention:**

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 0021h *unknown* | | | AX = *unknown* |
| STACK: | DWORD | -> *unknown* (8 bytes of data) | STACK unchanged |
| | WORD | *unknown* | |
| | WORD | *unknown* | |

**36-42 TKERNEL Function 0022h calling convention:**

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 0022h *unknown* | | | AX = *unknown* |
| STACK: | DWORD | -> *unknown* (8 bytes of data) | STACK unchanged |
| | DWORD | -> 4-byte buffer for results | |

**36-43 TKERNEL Function 0023h calling convention:**

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 0023h *unknown* | | | AX = *unknown* |
| STACK: | DWORD | -> *unknown* (8 bytes of data) | STACK unchanged |

**36-44 TKERNEL Function 0024h calling convention:**

| Registers at call: | | | Return Registers: |
|---|---|---|---|
| AX = 0024h set *unknown* | | | AX destroyed |
| STACK: | WORD | *unknown* | STACK unchanged |

### 36-45 TKERNEL Function 0025h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 0025h get *unknown* | AX = *unknown* (value set with func 0024h) |

### 36-46 TKERNEL Function 0026h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 0026h | crash: jumps randomly due to fencepost error |

### 36-47 TKERNEL "set DOS memory management functions" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = FFFFh<br>BX:SI -> FAR routine for allocating DOS memory<br>(called with AH=48h,BX=number of paragraphs to alloc; returns CF clear, AX=segment of allocated memory, or CF set on error)<br>CX:DI -> FAR routine for freeing DOS memory<br>(called with AH=49h,ES=segment of block to free; returns CF set on error,AX=error code) | none |

**Details:** Each of these pointers normally points at INT 21/RETF

### 36-48 Format of Function 001Eh data structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | unused |
| 02h | WORD | *unknown* |
| 04h | WORD | *unknown* |
| 06h | WORD | *unknown* |
| 08h | 2 BYTEs | unused |
| 0Ah | WORD | *unknown* |
| 0Ch | WORD | (call) *unknown*<br>(return) offset of this data structure (BUG?) |

## INT 2Fh Function FBA1h, Subfunction 0084h
## *UNINSTALL*

*TKERNEL*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FBA1h<br>BX = 0084h<br>ES:DI -> response buffer (see Table 36-49) | ES:DI buffer filled |

**See also:** Function FBA1h Subfunctions 0081h and 0084h

### 36-49 Format of response buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h 4 | BYTEs | signature "IABH" |
| 04h | WORD | success indicator<br>0001h failed (INT 2Fh hooked by another program)<br>unchanged if successful |
| 06h | WORD | segment of *unknown item* |
| 08h | WORD | segment of *unknown* memory block to free if nonzero |
| 0Ah | WORD | segment of *unknown* memory block to free if nonzero |

## CHAPTER 37

# *Other DOS Extenders*

Numerous other DOS extenders, in addition to those discussed in Chapters 33 through 36, do not have APIs extensive enough to warrant separate chapters. These DOS extenders (GO32, DBOS, AI Architects, and Windows/286), as well as some generic calls adopted by a variety of extenders, are described in this chapter.

## GO32

GO32.EXE is a DOS extender included as part of the 80386 port of the Free Software Foundation's GNU C/C++ compiler by DJ Delorie and distributed as DJGPP.

---

### INT 10h Function FFh *GO32.EXE*

***VIDEO EXTENSIONS***

| Registers at call: | Return Registers: |
|---|---|
| AH = FFh | *unknown* |
| AL = video mode (see Table 37-1) | |

**See also:** INT 21h Function FFh, Standard BIOS Function 00h (Chapter 8)

**37-1 GO32 video mode numbers:**

| | |
|---|---|
| 00h | 80x25 text |
| 01h | default text |
| 02h | CXxDX text |
| 03h | biggest text |
| 04h | 320x200 graphics |
| 05h | default graphics |
| 06h | CXxDX graphics |
| 07h | biggest non-interlaced graphics |
| 08h | biggest graphics |

---

### INT 21h Function FFh *GO32.EXE*

***DOS EXTENSIONS***

**Registers at call:**

AH = FFh
AL = function
    01h create file
    02h open file
    03h get file statistics
    04h get time of day
    05h set time of day

06h stat
07h system

**Return Registers:**
*unknown*

**See also:** INT 10h Function FFh

---

### INT 78h through INT 7Fh

*GO32.EXE*

#### *RELOCATED IRQ0 THROUGH IRQ7*

**Details:** These vectors are overwritten when GO32 starts but are not restored by early versions of the extender.

**Conflicts:** DBOS INT 78h and INT 79h (below), ADP-60 INT 78h and INT 79h (Chapter 10), TARGA.DEV INT 78h (Chapter 11), SCSILink INT 7Dh (Chapter 11), PRINDIR INT 7Ah and INT 7Ch (Chapter 15), XLOAD INT 7Eh (Chapter 41), AutoCAD (Chapter 66), Canon I-30F INT 7Ah (Chapter 66), Btrieve INT 7Bh (Chapter 66), REXX88PC INT 7Ch (Chapter 66), YTERM INT 7Dh-7Fh (Chapter 66), HyperPAD INT 7Dh (Chapter 66), Novell NetWare INT 7Ah (NI), IBM 3270 Workstation INT 7Ah (NI), Topware NOS INT 7Ah (NI), X.PC INT 7Ah (NI), Eicon Access INT 7Bh (NI), SK-UPPS INT 7Ch (NI); see Table 2-4 for INT 7Fh conflicts

**See also:** IRQ0 INT 08h (Chapter 3), INT 10h Function FFh, INT 21h Function FFh

## DBOS (University of Salford Extender)

---

### INT 78h

*DBOS DOS extender*

#### *API*

**Registers at call:**
AH = function (see Table 37-2)

**Return Registers:**
vary by function

**Details:** DBOS supports functions 00h through 50h; many of these functions provide a low-level debugging interface.

Command arguments are read from the calling program's PSP.

**Conflicts:** GO32 (above), ADP-60 (Chapter 10), AutoCAD (Chapter 66), TARGA.DEV (Chapter 11)

**See also:** DBOS INT 79h

**37-2 DBOS functions:**

| | |
|---|---|
| 00h | (PB) display 32-bit memory specified by command argument |
| 01h | (PT) display 32-bit instruction(s) specified by command arg |
| 02h | specify offset to subsequent PB and PT commands |
| 03h | switch to protected mode<br>DWORD following INT instruction points to map; protected-mode entry point is immediately following the DWORD.<br>This function never returns. |

**Details:** If bit 31 of the map address is set, only a stub was loaded by DOS and DBOS will load the entire program

| | |
|---|---|
| 04h | specify that subsequent load (Function 03h) should leave program in memory |
| 05h | release program from memory |
| 06h | set break point at address and option count from commandline |
| 07h | resume program execution |
| 08h | single-step program |
| 09h | set read/write breakpoint using 386 debug registers |
| 0Ah | set write breakpoint using 386 debug register |
| 0Bh | set memory byte to new value |
| 0Ch | display CPU registers |
| 0Dh | run until specified program address reached |
| 0Fh | print trace from program map |

| | |
|---|---|
| 10h | specify an offset using a map symbol |
| 14h | print memory without any offset |
| 18h | switch DBOS into/out of test mode (ON/OFF commandline args) |
| 1Dh | get address of real/protected-mode communication buffer.<br>**Return Registers:**<br>ES:BP -> comm buffer |
| 1Eh | set real-mode memory size (specify how much real-mode memory to leave free when running FTN77 programs) |
| 22h | uninstall DBOS |
| 24h | force DBOS to emulate coprocessor instructions |
| 26h | set named DBOS switches from commandline |
| 27h | reset named DBOS switched from commandline |
| 2Ah | set list of dynamic link libraries to contents of commandline file |
| 35h | specify that subsequent load (Function 03h) should stop at first instruction |

### INT 79h

*DBOS DOS Extender*

#### *MS WINDOWS SUPPORT*

**Purpose:** This interrupt is used by an MS Windows virtual device driver with which the DBOS extender communicates when run inside a Windows Enhanced Mode DOS box.

**Details:** Details were not available at the time of writing.

**Conflicts:** GO32 (above), AVATAR.SYS (Chapter 13), ADP-60 (Chapter 10), AutoCAD (Chapter 66)

**See also:** DBOS INT 78h

## Miscellaneous

This section describes assorted interrupts related to DOS extenders which either could not be identified accurately, or appeared in isolation.

### INT 21h Function 2Bh, Subfunction 4149h

*AI Architects*

#### *UNKNOWN APPLICATION - INSTALLATION CHECK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 2Bh | AL Ö FFh if installed |
| CX = 4149h ('AI') | |
| DX = 413Fh ('A?') | |

**Details:** Borland's TKERNEL makes this call.

### INT 2Fh Function 44h

*DOS Extender support (undocumented)*

#### *UNDETERMINED FUNCTIONS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 44h | *unknown* |
| AL = function (at least 0Bh, 15h, 17h) | |
| *others, if any, unknown* | |

**Details:** This function is called by Codeview for Windows.

**See also:** Function 86h (Chapter 57)

### INT 2Fh Function 46h

*Windows/286 DOS Extender (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 46h | *unknown* |
| AL = subfunction (03h,04h) | |

**Details:** These two subfunctions are called by MS Windows 3.0.

---

**INT 2Fh Function F100h** *DOS EXTENDER*

***GENERIC INSTALLATION CHECK***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = F100h | AL = FFh if DOS extender present<br>SI = 444Fh ("DO")<br>DI = 5358h ("SX") |

**Details:** This function is supported or soon to be supported by DOS extenders from Phar Lap, Rational, Ergo, and IGC.

**See also:** Ergo Function A1h (Chapter 34), Phar Lap Function ED00h (Chapter 33), TKERNEL Function FBA1h Subfunction 0081h (Chapter 36), DOS/16M INT 15h Function BF02h (Chapter 35)

## CHAPTER 38

# *EMS and XMS*

The earliest standard means of accessing memory beyond the one megabyte address space available on the original IBM PC was the Lotus-Intel-Microsoft Expanded Memory Specification. After the 286 became popular and difficulties arose in accessing extended memory without conflicts, Microsoft created the Extended Memory Specification to address those problems. Current versions of 386 memory managers such as those described in Chapters 39 through 42 support both EMS and XMS, as well as a number of other standards such as VCPI (Chapter 29), DPMI (Chapter 30), and VDS (Chapter 18).

## EMS

The Lotus/Intel/Microsoft Expanded Memory Specification 3.2 is the original interface to bank-switched memory allowing programs to use more memory than is addressable in real mode. Two additional variants of the specification (EEMS and LIM EMS 4.0) were later created to address shortcomings in the original standard.

Though not listed in each entry, the EMS INT 67h conflicts with some older software and some hard disk controllers; see Table 2-3.

---

### INT 67h Function 40h *LIM EMS*

### *GET MANAGER STATUS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 40h | AH = status (00h,80h,81h,84h) (see Table 38-1) |

**Details:** This call can be used only after establishing that the EMS driver is in fact present.

**See also:** QEMM Function 3Fh (Chapter 39), EMM386 Function FFA5h (Chapter 42)

**38-1 Values for EMS function status:**

| | |
|---|---|
| 00h | successful |
| 80h | internal error |
| 81h | hardware malfunction |
| 83h | invalid handle |
| 84h | undefined function requested by application |
| 85h | no more handles available |
| 86h | error in save or restore of mapping context |
| 87h | insufficient memory pages in system |
| 88h | insufficient memory pages available |
| 89h | zero pages requested |
| 8Ah | invalid logical page number encountered |
| 8Bh | invalid physical page number encountered |
| 8Ch | page-mapping hardware state save area is full |

| | |
|---|---|
| 8Dh | save of mapping context failed |
| 8Eh | restore of mapping context failed |
| 8Fh | undefined subfunction |
| 90h | undefined attribute type |
| 91h | feature not supported |
| 92h | successful, but a portion of the source region has been overwritten |
| 93h | length of source or destination region exceeds length of region allocated to either source or destination handle |
| 94h | conventional and expanded memory regions overlap |
| 95h | offset within logical page exceeds size of logical page |
| 96h | region length exceeds 1M |
| 97h | source and destination EMS regions have same handle and overlap |
| 98h | memory source or destination type undefined |
| 9Ah | specified DMA register set not supported |
| 9Bh | all DMA register sets currently allocated |
| 9Ch | alternate DMA sets not supported |
| 9Dh | undefined or unallocated DMA register set |
| 9Eh | dedicated DMA channels not supported |
| 9Fh | specified dedicated DMA channel not supported |
| A0h | no such handle name |
| A1h | a handle found had no name, or duplicate handle name |
| A2h | attempted to wrap around 1M conventional address space |
| A3h | source array corrupted |
| A4h | operating system denied access |

---

### INT 67h Function 41h
*LIM EMS*

### *GET PAGE FRAME SEGMENT*

**Registers at call:**
AH = 41h

**Return Registers:**
AH = status (see also Table 38-1)
00h function successful
BX = segment of page frame

**See also:** Functions 58h and 68h

---

### INT 67h Function 42h
*LIM EMS*

### *GET NUMBER OF PAGES*

**Registers at call:**
AH = 42h

**Return Registers:**
AH = 00h function successful
BX = number of unallocated pages
DX = total number of pages
AH = error code (see Table 38-1)

**BUG:** The DOS 6.0 EMM386.EXE causes a system lock-up or reboot if in AUTO mode when this call is made; use Function 46h to ensure that EMM386 is ON before making this call.

**See also:** TaskMAX INT 2Fh Function 2702h (Chapter 47)

---

### INT 67h Function 43h
*LIM EMS*

### *GET HANDLE AND ALLOCATE MEMORY*

**Registers at call:**
AH = 43h
BX = number of logical pages to allocate

**Return Registers:**
AH = status
(00h,80h,81h,84h,85h,87h,88h,89h)
(see Table 38-1)
DX = handle if AH=00h

**See also:** Function 45h

**INT 67h Function 44h** *LIM EMS*

***MAP MEMORY***

**Registers at call:**
AH = 44h
AL = physical page number (0-3)
BX = logical page number
DX = handle

**Return Registers:**
AH = status (00h,80h,81h,83h,84h,8Ah,8Bh) (see Table 38-1)

**See also:** Function 69h

---

**INT 67h Function 45h** *LIM EMS*

***RELEASE HANDLE AND MEMORY***

**Registers at call:**
AH = 45h
DX = EMM handle

**Return Registers:**
AH = status (00h,80h,81h,83h,84h,86h) (see Table 38-1)

**See also:** Function 43h

---

**INT 67h Function 46h** *LIM EMS*

***GET EMM VERSION***

**Registers at call:**
AH = 46h

**Return Registers:**
AH = status (00h,80h,81h,84h) (see Table 38-1)
AL = EMM version number if AH=00h

---

**INT 67h Function 47h** *LIM EMS*

***SAVE MAPPING CONTEXT***

**Registers at call:**
AH = 47h
DX = handle

**Return Registers:**
AH = status (00h,80h,81h,83h,84h,8Ch-8Eh) (see Table 38-1)

**See also:** Function 48h

---

**INT 67h Function 48h** *LIM EMS*

***RESTORE MAPPING CONTEXT***

**Registers at call:**
AH = 48h
DX = handle

**Return Registers:**
AH = status (00h,80h,81h,83h,84h,8Eh) (see Table 38-1)

**See also:** Function 47h

---

**INT 67h Function 49h** *LIM EMS*

***reserved - GET I/O PORT ADDRESSES***

**Registers at call:**
AH = 49h

**Return Registers:**
*unknown*

**Details:** This function is defined in EMS 3.0, but undocumented in EMS 3.2.

---

**INT 67h Function 4Ah** *LIM EMS*

***reserved - GET TRANSLATION ARRAY***

**Registers at call:**
AH = 4Ah

**Return Registers:**
*unknown*

**Details:** This function is defined in EMS 3.0, but undocumented in EMS 3.2.

---

**INT 67h Function 4Bh** *LIM EMS*

***GET NUMBER OF EMM HANDLES***

**Registers at call:**
AH = 4Bh

**Return Registers:**
AH = status (00h,80h,81h,83h,84h) (see Table 38-1)
BX = number of EMM handles if AH=00h

LIM EMS

## INT 67h Function 4Ch
### *GET PAGES OWNED BY HANDLE*

**Registers at call:**
AH = 4Ch
DX = EMM handle

**Return Registers:**
AH = status (00h,80h,81h,83h,84h) (see Table 38-1)
BX = number of logical pages if AH=00h

**See also:** Function 4Dh

---

LIM EMS

## INT 67h Function 4Dh
### *GET PAGES FOR ALL HANDLES*

**Registers at call:**
AH = 4Dh
ES:DI -> array to receive information

**Return Registers:**
AH = status (00h,80h,81h,84h) (see Table 38-1)
---**if AH=00h**---
BX = number of active EMM handles
array filled with 2-word entries, consisting of a handle and the number of pages allocated to that handle

**See also:** Function 4Ch

---

LIM EMS

## INT 67h Function 4Eh
### *GET OR SET PAGE MAP*

**Registers at call:**
AH = 4Eh
AL = subfunction
- 00h if getting mapping registers
- 01h if setting mapping registers
- 02h if getting and setting mapping registers at once
- 03h if getting size of page-mapping array

DS:SI -> array holding information (subfunctions 01h/02h)
ES:DI -> array to receive information (subfunctions 00h/02h)

**Return Registers:**
AH = status (00h,80h,81h,84h,8Fh,A3h) (see also Table 38-1)
if successful
- AL = bytes in page-mapping array (subfunction 03h only)
- array pointed to by ES:DI receives mapping info (subfunctions 00h/02h)

**Details:** This function was designed to be used by multitasking operating systems and should not ordinarily be used by appplication software.

MD386 returns the size of the page-mapping array in AX instead of AL.

**See also:** Function 4Fh

---

LIM EMS 4.0

## INT 67h Function 4Fh
### *GET/SET PARTIAL PAGE MAP*

**Registers at call:**
AH = 4Fh
AL = subfunction
- 00h get partial page map
  - DS:SI -> structure containing list of segments whose mapping contexts are to be saved
  - ES:DI -> array to receive page map
- 01h set partial page map
  - DS:SI -> structure containing saved partial page map
- 02h get size of partial page map
  - BX = number of mappable segments in the partial map to be saved

**Return Registers:**
AH = status (00h,80h,81h,84h,8Bh,8Fh,A3h) (see Table 38-1)
AL = size of partial page map for subfunction 02h

**See also:** Function 4Eh

## INT 67h Function 50h

*LIM EMS 4.0*

### *MAP/UNMAP MULTIPLE HANDLE PAGES*

**Registers at call:**

AH = 50h
AL = subfunction
    00h use physical page numbers
    01h use segment addresses
DX = handle
CX = number of entries in array
DS:SI -> mapping array (see Table 38-2)

**Return Registers:**

AH = status (00h,80h,81h,83h,84h,8Ah,8Bh,8Fh)
(see Table 38-1)

**See also:** Function 40h

**38-2 Format of mapping array entry:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | logical page number or FFFFh to unmap physical page |
| 02h | WORD | physical page number or segment address |

## INT 67h Function 51h

*LIM EMS 4.0*

### *REALLOCATE PAGES*

**Registers at call:**

AH = 51h
DX = handle
BX = number of pages to be allocated to handle

**Return Registers:**

BX = actual number of pages allocated to handle
AH = status (00h,80h,81h,83h,84h,87h,88h)
(see Table 38-1)

## INT 67h Function 52h

*LIM EMS 4.0*

### *GET/SET HANDLE ATTRIBUTES*

**Registers at call:**

AH = 52h
DX = handle
AL = subfunction
    00h get handle attributes
    01h set handle attributes
        BL = new attribute
        (see returned AL)
    02h get attribute capability

**Return Registers:**

AH = status (00h,80h,81h,83h,84h,8Fh-91h)
(see Table 38-1)

(subfunction 00h:)
AL = attribute
    00h handle is volatile
    01h handle is nonvolatile

(subfunction 02h:)
AL = attribute capability
    00h only volatile handles supported
    01h both volatile and non-volatile supported

**See also:** Function 53h

## INT 67h Function 53h

*LIM EMS 4.0*

### *GET/SET HANDLE NAME*

**Registers at call:**

AH = 53h
DX = handle
AL = subfunction
    00h get handle name
        ES:DI -> 8-byte buffer for handle name
    01h set handle name
        DS:SI -> 8-byte handle name

**Return Registers:**

AH = status (00h,80h,81h,83h,84h,8Fh,A1h)
(see Table 38-1)

**See also:** Function 52h

## INT 67h Function 54h

LIM EMS 4.0

### *GET HANDLE DIRECTORY*

**Registers at call:**
AH = 54h
AL = subfunction
  00h get handle directory
    ES:DI -> buffer for handle directory (see Table 38-3)
  01h search for named handle
    DS:SI -> 8-byte name
  02h get total number of handles

**Return Registers:**
AH = status (00h,80h,81h,84h,8Fh,A0h,A1h) (see Table 38-1)
AL = number of entries in handle directory (subfunction 00h)
DX = value of named handle (subfunction 01h)
BX = total number of handles (subfunction 02h)

**38-3 Format of handle directory entry:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | handle |
| 02h | 8 BYTEs | handle's name |

## INT 67h Function 55h

LIM EMS 4.0

### *ALTER PAGE MAP AND JUMP*

**Registers at call:**
AH = 55h
AL = subfunction
  00h physical page numbers provided by caller
  01h segment addresses provided by caller
DX = handle
DS:SI -> structure containing map and jump address

**Return Registers:**
(at target address unless error)
AH = status (00h,80h,81h,83h,84h,8Ah,8Bh,8Fh) (see Table 38-1)

**See also:** Function 56h

## INT 67h Function 56h

LIM EMS 4.0

### *ALTER PAGE MAP AND CALL*

**Registers at call:**
AH = 56h
AL = subfunction
  00h physical page numbers provided by caller
    DX = handle
    DS:SI -> structure containing page map and call address
  01h segment addresses provided by caller
    DX = handle
    DS:SI -> structure containing page map and call address
  02h get page map stack space required

**Return Registers:**
(If successful, the target address is called. Use a RETF to return and restore mapping context)
AH = status (00h,80h,81h,83h,84h,8Ah,8Bh,8Fh) (see Table 38-1)
BX = stack space required

**See also:** Function 55h

## INT 67h Function 57h
LIM EMS 4.0

### MOVE/EXCHANGE MEMORY REGION

**Registers at call:**
AH = 57h
AL = subfunction
  00h move memory region
  01h exchange memory region
DS:SI -> structure describing source and destination (see Table 38-4)

**Return Registers:**
AH = status (00h,80h,81h,83h,84h,8Ah,8Fh,92h-98h,A2h (see Table 38-1)

**Details:** The source and destination may overlap for a move, in which case the copy direction is chosen such that the destination receives an intact copy of the source region.

**38-4 Format of EMS copy data:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | region length in bytes |
| 04h | BYTE | source memory type (00h conventional, 01h expanded) |
| 05h | WORD | source handle (0000h if conventional memory) |
| 07h | WORD | source initial offset (within page if EMS, segment if conventional) |
| 09h | WORD | source initial segment (conventional) or logical page (EMS) |
| 0Bh | BYTE | destination memory type (00h conventional, 01h expanded) |
| 0Ch | WORD | destination handle |
| 0Eh | WORD | destination initial offset |
| 10h | WORD | destination initial segment or page |

## INT 67h Function 58h
LIM EMS 4.0

### GET MAPPABLE PHYSICAL ADDRESS ARRAY

**Registers at call:**
AH = 58h
AL = subfunction
  00h get mappable physical address array
    ES:DI -> buffer to be filled with array
  01h get number of entries in m.p.a. array

**Return Registers:**
AH = status (00h,80h,81h,84h,8Fh) (see Table 38-1)
CX = number of entries in array

**Details:** The returned array for subfunction 00h is filled in physical segment address order.

**38-5 Format of mappable physical address entry:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | physical page segment |
| 02h | WORD | physical page number |

## INT 67h Function 59h
LIM EMS 4.0

### GET EXPANDED MEMORY HARDWARE INFORMATION

**Registers at call:**
AH = 59h
AL = subfunction
  00h get hardware configuration array
    ES:DI -> buffer to be filled with array (see Table 38-6)
  01h get unallocated raw page count

**Return Registers:**
AH = status (00h,80h,81h,84h,8Fh,A4h) (see Table 38-1)
BX = unallocated raw pages
DX = total raw pages

**Details:** Subfunction 00h is for use by operating systems only, and can be enabled or disabled at any time by the operating system.

**38-6 Format of hardware configuration array:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | size of raw EMM pages in paragraphs |
| 02h | WORD | number of alternate register sets |
| 04h | WORD | size of mapping-context save area in bytes |
| 06h | WORD | number of register sets assignable to DMA |
| 08h | WORD | DMA operation type<br>0000h DMA with alternate register sets<br>0001h only one DMA register set |

---

### INT 67h Function 5Ah

*LIM EMS 4.0*

#### *ALLOCATE STANDARD/RAW PAGES*

**Registers at call:**

AH = 5Ah
AL = subfunction
    00h allocate standard pages
    01h allocate raw pages
BX = number of pages to allocate

**Return Registers:**

AH = status (00h,80h,81h,84h,85h,87h,88h,8Fh)
  (see Table 38-1)
DX = handle

---

### INT 67h Function 5Bh

*LIM EMS 4.0*

#### *ALTERNATE MAP REGISTER SET*

**Registers at call:**

AH = 5Bh
AL = subfunction

    00h get alternate map register set

    01h set alternate map register set
        BL = new alternate map register set number
        ES:DI -> map register context save area if BL=00h
    02h get alternate map save array size

    03h allocate alternate map register set

    04h deallocate alternate map register set
        BL = number of alternate map register set

**Return Registers:**

AH = status
(00h,80h,81h,84h,8Fh,9Ah-9Dh,A3h,A4h)
  (see Table 38-1)

BL = current active alternate map register set number
ES:DI -> map register context save area if BL=00h

DX = array size in bytes

BL = number of map register set; 00h = not supported

**Details:** This function is for use by operating systems only, and can be enabled or disabled at any time by the operating system.

---

### INT 67h Function 5Bh

*LIM EMS 4.0*

#### *ALTERNATE MAP REGISTER SET - DMA REGISTERS*

**Registers at call:**

AH = 5Bh
AL = subfunction

    05h allocate DMA register set

    06h enable DMA on alternate map register set

BL = DMA register set number
DL = DMA channel number
07h disable DMA on alternate map register set
BL = DMA register set number
08h deallocate DMA register set
BL = DMA register set number

**Return Registers:**
AH = status (00h,80h,81h,84h,8Fh,9Ah-9Fh,A3h,A4h) (see Table 38-1)

BL = DMA register set number, 00h if not supported

**Details:** This function is for use by operating systems only, and can be enabled or disabled at any time by the operating system.

---

### INT 67h Function 5Ch
*LIM EMS 4.0*

### *PREPARE EXPANDED MEMORY HARDWARE FOR WARM BOOT*

**Registers at call:**
AH = 5Ch

**Return Registers:**
AH = status (00h,80h,81h,84h) (see Table 38-1)

---

### INT 67h Function 5Dh
*LIM EMS 4.0*

### *ENABLE/DISABLE OS FUNCTION SET FUNCTIONS*

**Registers at call:**
AH = 5Dh
AL = subfunction
00h enable OS Function Set
01h disable OS Function Set
02h return access key (resets memory manager, returns access key at next invocation)
BX,CX = access key returned by first invocation

**Return Registers:**
AH = status (00h,80h,81h,84h,8Fh,A4h) (see Table 38-1)
BX,CX = access key, returned only on first invocation of function

---

### INT 67h Function 60h
*EEMS*

### *GET PHYSICAL WINDOW ARRAY*

**Registers at call:**
AH = 60h
ES:DI -> buffer

**Return Registers:**
AH = status (see also Table 38-1)
AL = number of entries
buffer at ES:DI filled

---

### INT 67h Function 61h
*EEMS*

### *GENERIC ACCELERATOR CARD SUPPORT*

**Registers at call:**
AH = 61h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** This function can be used by an accelerator card manufacturer to flush the RAM cache, ensuring that the cache accurately reflects what the processor would see without the cache.

---

### INT 67h Function 68h
*EEMS*

### *GET ADDRESSES OF ALL PAGE FRAMES IN SYSTEM*

**Registers at call:**
AH = 68h
ES:DI -> buffer

**Return Registers:**
AH = status (see also Table 38-1)
AL = number of entries
buffer at ES:DI filled

**Details:** This function is equivalent to LIM 4.0 Function 58h.

---

### INT 67h Function 69h

*EEMS*

### *MAP PAGE INTO FRAME*

**Registers at call:**
AH = 69h
AL = frame number
BX = page number
DX = handle

**Return Registers:**
AH = status (see also Table 38-1)

**Details:** This function is similar to EMS Function 44h.

**See also:** Functions 44h, 50h, and 6Ah

---

### INT 67h Function 6Ah

*EEMS*

### *PAGE MAPPING*

**Registers at call:**
AH = 6Ah
AL = subfunction

- 00h save partial page map
  - CH = first page frame
  - CL = number of frames
  - ES:DI -> buffer which is to be filled
- 01h restore partial page map
  - CH = first page frame
  - CL = number of frames
  - DI:SI -> previously saved page map
- 02h save and restore partial page map
  - CH = first page frame
  - CL = number of frames
  - ES:DI = buffer for current page map
  - DI:SI = new page map
- 03h get size of save array
  - CH = first page frame
  - CL = number of frames
- 04h switch to standard map register setting
- 05h switch to alternate map register setting
- 06h deallocate pages mapped to frames in conventional memory
  - CH = first page frame
  - CL = number of frames

**Return Registers:**
AH = status (see also Table 38-1)
AL = size of array in bytes

**Details:** This function is similar to EMS Function 4Eh, except that a subrange of pages can be specified.

**See also:** Function 69h

## Extended Memory Specification (XMS)

The Extended Mmeory Specification gives accesss to extended memory and noncontiguous, non-EMS memory above the 640K point. It provides controlled access to extended memory, unlike BIOS INT 15h Function 87h and 88h (Chapter 4), thus permitting applications to share extended memory without conflicts. XMS is provided by Microsoft's HIMEM.SYS and most 386 memory managers.

## INT 2Fh Function 1610h
*XMS v1.x only*
*Obsolete*

### *GET DRIVER ADDRESS*

**Registers at call:**
AX = 1610h
details unavailable

**Return Registers:**
not known

**Details:** This function and Function 1600h were only used in XMS version 1 and are now obsolete. Use Functions 4300h and 4310h instead.

**See also:** Function 4310h, Windows/286 Function 1600h (Chapter 43)

## INT 2Fh Function 4300h
*XMS v2+*

### *INSTALLATION CHECK*

**Registers at call:**
AX = 4300h

**Return Registers:**
AL = 80h if XMS driver installed

**Details:** This installation check DOES NOT follow the format used by other software.

**See also:** Function 4310h

## INT 2Fh Function 4310h
*XMS v2+*

### *GET DRIVER ADDRESS*

**Registers at call:**
AX = 4310h

**Return Registers:**
ES:BX -> driver entry point (see Tables 38-7 to 38-29)

**Details:** HIMEM.SYS v2.77 chains to the previous handler if AH is not 00h or 10h.

HIMEM.SYS requires at least 256 bytes free stack space.

The XMS driver need not implement functions 10h through 12h to be considered compliant with the standard.

QEMM-386 (v5.11 only) also supports driver functions 34h and 44h, but no information on these undocumented calls was available.

**See also:** Function 4300h, RM386 Function 4310h (Chapter 41)

### 38-7 XMS "Get XMS version number" calling convention:

**Registers at call:**
AH = 00h Get XMS version number

**Return Registers:**
AX = XMS version (in BCD, AH=major, AL=minor)
BX = internal revision number
DX = HMA status
  0001h if HMA (1M to 1M + 64K) exists
  0000h if HMA does not exist

### 38-8 XMS "Request High Memory Area" calling convention:

**Registers at call:**
AH = 01h
DX = memory in bytes (for TSR or device drivers)
  FFFFh if application program

**Return Registers:**
AX = status
  0001h success
  0000h failure
    BL = error code (80h,81h,90h,91h,92h) (see Table 38-30)

### 38-9 XMS "Release High Memory Area" calling convention:

**Registers at call:**
AH = 02h

**Return Registers:**
AX = status
  0001h success
  0000h failure
    BL = error code (80h,81h,90h,93h) (see Table 38-30)

### 38-10 XMS "Global enable A20" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 03h | AX = status |
| | 0001h success |
| | 0000h failure |
| | BL = error code (80h,81h,82h) (see Table 38-30) |

### 38-11 XMS "Global disable A20" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 04h | AX = status |
| | 0001h success |
| | 0000h failure |
| | BL = error code (80h,81h,82h,94h) (see Table 38-30) |

### 38-12 XMS "Local enable A20" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 05h | AX = status |
| | 0001h success |
| | 0000h failure |
| | BL = error code (80h,81h,82h) (see Table 38-30) |

**Details:** Use this function for direct access to extended memory.

### 38-13 XMS "Local disable A20" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 06h | AX = status |
| | 0001h success |
| | 0000h failure |
| | BL = error code (80h,81h,82h,94h) (see Table 38-30) |

### 38-14 XMS "Query A20 state" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 07h | AX = status |
| | 0001h enabled |
| | 0000h disabled |
| | BL = error code (00h,80h,81h) (see Table 38-30) |

### 38-15 XMS "Query free extended memory, not counting HMA" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 08h | AX = size of largest extended memory block in KB |
| BL = 00h (some implementations leave BL unchanged on success) | DX = total extended memory in KB |
| | BL = error code (00h,80h,81h,A0h) (see Table 38-30) |

### 38-16 XMS "Allocate extended memory block" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 09h | AX = status |
| DX = Kbytes needed | 0001h success |
| | DX = handle for memory block |
| | 0000h failure |
| | BL = error code (80h,81h,A0h) (see Table 38-30) |

### 38-17 XMS "Free extended memory block" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 0Ah | AX = status |
| DX = handle of block to free | 0001h success |
| | 0000h failure |
| | BL = error code (80h,81h,A2h,ABh) (see Table 38-30) |

## INT 2Fh Function 4310h

### 38-18 XMS "Move extended memory block" calling convention:

**Registers at call:**
AH = 0Bh
DS:SI -> EMM structure (see Table 38-31)

**Return Registers:**
AX = status
0001h success
0000h failure
BL = error code (80h-82h,A3h-A9h) (see Table 38-30)

**Details:** If either handle is 0000h, the corresponding offset is considered to be an absolute segment:offset address in directly addressable memory.

### 38-19 XMS "Lock extended memory block" calling convention:

**Registers at call:**
AH = 0Ch
DX = handle of block to lock

**Return Registers:**
AX = status
0001h success
DX:BX = 32-bit linear address of locked block
0000h failure
BL = error code (80h,81h,A2h,ACh,ADh) (see Table 38-30)

**Details:** MS Windows 3.x reject this function for handles allocated after Windows started.

### 38-20 XMS "Unlock extended memory block" calling convention:

**Registers at call:**
AH = 0Dh
DX = handle of block to unlock

**Return Registers:**
AX = status
0001h success
0000h failure
BL = error code (80h,81h,A2h,AAh) (see Table 38-30)

### 38-21 XMS "Get handle information" calling convention:

**Registers at call:**
AH = 0Eh
DX = handle for which to get info

**Return Registers:**
AX = status
0001h success
BH = block's lock count
BL = number of free handles left
DX = block size in KB
0000h failure
BL = error code (80h,81h,A2h) (see Table 38-30)

**BUG:** MS Windows 3.10 acts as though unallocated handles are in use.
**Details:** MS Windows 3.00 has problems with this call.

### 38-22 XMS "Reallocate extended memory block" calling convention:

**Registers at call:**
AH = 0Fh
DX = handle of block
BX = new size of block in KB

**Return Registers:**
AX = status
0001h success
0000h failure
BL = error code (80h,81h,A0h-A2h,ABh) (see Table 38-30)

### 38-23 XMS "Request upper memory block" calling convention:

**Registers at call:**
AH = 10h
DX = size of block in paragraphs

**Return Registers:**
AX = status
0001h success
BX = segment address of UMB
DX = actual size of block
0000h failure
BL = error code (80h,B0h,B1h) (see Table 38-30
DX = largest available block

### 38-24 XMS "Release upper memory block" calling convention:

**Registers at call:**
AH = 11h
DX = segment address of UMB to release

**Return Registers:**
AX = status
  0001h success
  0000h failure
    BL = error code (80h,B2h) (see Table 38-30)

### 38-25 XMS v3.0 "Reallocate upper memory block" calling convention:

**Registers at call:**
AH = 12h
DX = segment address of UMB to resize
BX = new size of block in paragraphs

**Return Registers:**
AX = status
  0001h success
  0000h failure
    BL = error code (80h,B0h,B2h) (see Table 38-30)
    DX = maximum available size (RM386)

### 38-26 XMS v3.0 "Query free extended memory" calling convention:

**Registers at call:**
AH = 88h

**Return Registers:**
EAX = largest block of extended memory, in KB
BL = status
  00h success
  80h not implemented (i.e. on a 286 system)
  81h VDISK detected
  A0h all extended memory allocated
ECX = physical address of highest byte of memory (valid even on error codes 81h and A0h)
EDX = total Kbytes of extended memory (0 if status A0h)

### 38-27 XMS v3.0 "Allocate any extended memory" calling convention:

**Registers at call:**
AH = 89h
EDX = Kbytes needed

**Return Registers:**
AX = status
  0001h success
    DX = handle for allocated block (free with Function 0Ah)
  0000h failure
    BL = status (80h,81h,A0h,A1h,A2h) (see Table 38-30)

### 38-28 XMS v3.0 "Get extended EMB handle information" calling convention:

**Registers at call:**
AH = 8Eh
DX = handle

**Return Registers:**
AX = status
  0001h success
    BH = block's lock count
    CX = number of free handles left
    EDX = block size in KB
  0000h failure
    BL = status (80h,81h,A2h) (see Table 38-30)

**BUG:** DOS 6.0 HIMEM.SYS leaves CX unchanged.

### 38-29 XMS v3.0 "Reallocate any extended memory block" calling convention:

**Registers at call:**
AH = 8Fh
DX = unlocked handle
EBX = new size in KB

**Return Registers:**
AX = status
  0001h success
  0000h failure
    BL = status (80h,81h,A0h-A2h,ABh) (see Table 38-30)

**BUG:** HIMEM v3.03-3.07 crash on an 80286 machine if any of the 8Xh functions are called.

## INT 2Fh Function 4310h

### 38-30 Error codes returned in BL:

| | |
|---|---|
| 00h | successful |
| 80h | function not implemented |
| 81h | Vdisk was detected |
| 82h | an A20 error occurred |
| 8Eh | a general driver error |
| 8Fh | unrecoverable driver error |
| 90h | HMA does not exist |
| 91h | HMA is already in use |
| 92h | DX is less than the /HMAMIN= parameter |
| 93h | HMA is not allocated |
| 94h | A20 line still enabled |
| A0h | all extended memory is allocated |
| A1h | all available extended memory handles are allocated |
| A2h | invalid handle |
| A3h | source handle is invalid |
| A4h | source offset is invalid |
| A5h | destination handle is invalid |
| A6h | destination offset is invalid |
| A7h | length is invalid |
| A8h | move has an invalid overlap |
| A9h | parity error occurred |
| AAh | block is not locked |
| ABh | block is locked |
| ACh | block lock count overflowed |
| ADh | lock failed |
| B0h | only a smaller UMB is available |
| B1h | no UMB's are available |
| B2h | UMB segment number is invalid |

### 38-31 Format of EMM structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | number of bytes to move (must be even) |
| 04h | WORD | source handle |
| 06h | DWORD | offset into source block |
| 0Ah | WORD | destination handle |
| 0Ch | DWORD | offset into destination block |

**Notes:** If the source and destination overlap, only forward moves (source base less than destination base) are guaranteed to work properly.

If either handle is zero, the corresponding offset is interpreted as a real-mode address referring to memory directly addressable by the processor.

## CHAPTER 39

# Quarterdeck QEMM-386

Quarterdeck Office Systems' QEMM-386 is one of the most popular third-party memory managers. When run under QEMM, Quarterdeck's DESQview multitasker becomes DESQview/386 through the use of various hooks in the QEMM private API. QEMM was the first memory manager to support the VCPI specification, as it was formulated by Quarterdeck and Phar Lap; a portion of the QEMM private API (Tables 39-18 through 39-28) mirrors the VCPI functionality and is the likely ancestor of VCPI.

---

### INT 21h Function 4402h

*Quarterdeck (undocumented)*

### *QEMM-386 v5+ - GET API ENTRY POINT*

| Registers at call: | Return Registers: |
|---|---|
| AX = 4402h | CF clear if successful |
| BX = file handle for device "QEMM386$" | buffer filled (see Tables 39-1 through 39-81 for entry point parameters) |
| CX = 0004h | CF set on error |
| DS:DX -> DWORD buffer for API entry point | AX = error code (01h,05h,06h,0Dh) (see Table 24-2) |

**Details:** Quarterdeck recently (June 1993) documented this function, but the documentation incorrectly states that it is only available for QEMM 6+.

**See also:** INT 2Fh Function D201h Subfunction 5145h, INT 67h Function 3Fh, HOOKROM Function 4402h (Chapter 63)

---

### INT 2Fh Function D201h, Subfunction 5145h

*Quarterdeck RPCI*

### *QEMM v5.0+ - INSTALLATION CHECK*

| Registers at call: | Return Registers: |
|---|---|
| AX = D201h | BX = 4F4Bh ("OK") |
| BX = 5145h ("QE") | ES:DI -> QEMM API entry point (see Tables 39-1 to 39-82) |
| CX = 4D4Dh ("MM") | |
| DX = 3432h ("42") | |

**Details:** Resident Quarterdeck programs will search for a free AH value from D2h through FFh, then C0h through D1h; see Function D200h in Chapter 63 for details.

This call is not available under QEMM v6.00 unless Windows3 support has been disabled with the NW3 switch to QEMM386.SYS.

**Conflicts:** PCL-838 (Chapter 66); see also Table 2-2

**See also:** INT 67h Function 3Fh, RPCI Function D200h (Chapter 63),Function D201h Subfunction 4849h (Chapter 63)

---

### INT 4Bh Function 810Dh

QEMM-386

***BUG***

**Registers at call:**
AX = 810Dh

**Return Registers:**
crash

**Details:** The code in QEMM v5.11 and 6.00 jumps to an invalid location on this call.

---

### INT 67h Function 3Fh, Subfunction 5145h

*QEMM-386 v4.23+*
*(undocumented)*

***INSTALLATION CHECK***

**Registers at call:**
AH = 3Fh
CX = 5145h ("QE")
DX = 4D4Dh ("MM")

**Return Registers:**
AH = 00h if installed
ES:DI -> QEMM API entry point

**Details:** If no other program has hooked INT 67h, an alternate installation check is to test for the string "QUARTERDECK EXPANDED MEMORY MANAGER 386" at offset 14h in the INT 67h handler's segment; the word at offset 12h contains the offset in the handler's segment of the API entry point.

Although this function is still undocumented, Quarterdeck has recently documented two alternate methods for determining the QEMM API entry point, as well as several of the API functions.

MICEMM (Micronics Expanded Memory Manager) versions 2.0C and 4D support the alternate QEMM installation check and entry point functions 00h, 02h, and 03h; version 4D only provides the signature string if the commandline argument "DV" is provided.

386MAX V6.01 responds to this call, but DESQview 2.42 does not recognize the returned entry point as providing QEMM's capabilities because

a) only functions 0Ch (different from QEMM 0Ch) and 1000h-1009h are supported,
b) status is returned as for EMS functions, not QEMM functions
c) the protected-mode entry point returned by function 1000h only supports functions 0Ch, 1004h, 1005h, and 100Ah.

The string check mentioned above is not supported by 386MAX.

**See also:** QEMM INT 21h Function 4402h, Netroom Function 5BF0h (Chapter 41), QMAPS Function DDh (Chapter 42), EMM386 Function FFA5h (Chapter 42), QEXT INT 15h Function 11DEh (Chapter 63), Memory Manager INT 21h Function 4402h Subfunction 01h (Chapter 42), 386MAX INT 21h Function 4402h (Chapter 40), Quarterdeck INT 2Fh Function D201h Subfunction 5145h (Chapter 63)

#### 39-1 QEMM "get state" (documented) calling convention:

**Registers at call:**
AH = 00h

**Return Registers:**
CF clear
AL = QEMM state
bit 0 set if QEMM turned OFF
bit 1 set if in "Auto" mode

#### 39-2 QEMM "set state" (documented) calling convention:

**Registers at call:**
AH = 01h
AL = new state
bit 0 set: place QEMM in OFF state

**Return Registers:**
CF clear if successful
CF set on error

#### 39-3 QEMM Function 02h calling convention:

**Registers at call:**
AH = 02h get *unknown*

**Return Registers:**
CF clear
AX = segment of *unknown* data structure (see Table 39-4)

### 39-4 QEMM Function 02h data structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | page table entry for *unknown* page table |
| | | *others, if any, unknown* |

### 39-5 QEMM "get version" (documented) calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 03h | CF clear |
| | AX = BX = version in BCD |

**Details:** The most recent official docs state that the version is returned in both AX and BX; older documentation only mentions BX.

MICEMM returns AX=0001h, BX unchanged.

### 39-6 QEMM "allocate 4K page and set AUTO/ON mode" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 04h | CF clear if successful |
| | DX = page number of a 4K page |
| | CF set if unable to allocate page |

**Details:** QEMM's mode remains unchanged if not AUTO/OFF.

### 39-7 QEMM "free 4K page and turn QEMM off" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 05h | CF clear |
| DX = page number returned by Function 04h (see Table 39-6) | |

**Details:** QEMM's mode remains unchanged if not AUTO/ON.

### 39-8 QEMM "make new mapping context" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 06h *make new mapping context* | CF clear |
| DX = page number of 4K page to hold page table. | |

**Details:** This function copies the page table into given page and then sets an *unknown* page table entry to point at the copy.

### 39-9 QEMM "get mapping context" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 07h | CF clear |
| | DX = page number of page table for current mapping context |

### 39-10 QEMM "set mapping context" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 08h | CF clear |
| DX = linear page number of page table | |

### 39-11 QEMM "get linear page number for page table entry" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 09h | CF clear |
| CX = page table index | DX = linear page number |

### 39-12 QEMM "set linear page number for page table entry" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 0Ah | CF clear |
| CX = page table index | |
| DX = linear page number | |

**39-13 QEMM "map 4K pages" calling convention:**

**Registers at call:**
AH = 0Bh
BX = number of pages
CX = first page number (must be 0100h to allocate HMA)
DX = EMS handle (memory belonging to EMS handle will be mapped into the address space beginning with the first page allocated to the handle)

**Return Registers:**
AH = 00h

**39-14 QEMM "get available memory" calling convention:**

**Registers at call:**
AH = 0Ch

**Return Registers:**
CF clear
BX = 0001h
CX = *total 4K pages*
DX = number of 4K pages free

**39-15 QEMM Function 0Dh calling convention:**

**Registers at call:**
AH = 0Dh *unknown* (related to callbacks)
AL = 00h/01h/02h *unknown*

**Return Registers:**
CF clear

**39-16 QEMM Function 0Eh calling convention:**

**Registers at call:**
AH = 0Eh set *unknown* callbacks
DS:BX -> FAR routine for *unknown*
ES:DX -> FAR routine for *unknown*

**Return Registers:**
CF clear

**Details:** The DS:BX callback should return BX=*unknown*; ES:DX is called with BX=*unknown*, and should set the *unknown* from which the other handler read the value of BX. BH and BL appear to be separate values.

**39-17 QEMM "unmap 4K pages" calling convention:**

**Registers at call:**
AH = 0Fh
CX = first page number
DX = number of pages

**Return Registers:**
CF clear
AL = 00h/01h if *unknown*

**Details:** If CX=0100h and DX=0010h, the HMA is remapped to simulate a disabled A20.

**39-18 QEMM "get protected-mode interface" calling convention:**

**Registers at call:**
AX = 1000h
DS:SI -> 16-byte buffer for two GDT entries
ES:DI -> buffer for 4K page table

**Return Registers:**
CF clear
EAX = offset of protected-mode API entry point
DS:SI buffer filled with two GDT descriptors; *first is QEMM code segment, second is data*
ES:DI buffer filled with 4K page table
DI points to first unused page table entry

**See also:** INT 67h Function DE01h (Chapter 29)

**39-19 QEMM "get CPU debug registers" calling convention:**

**Registers at call:**
AX = 1001h
ES:DI -> buffer for debug registers (8 DWORDs)

**Return Registers:**
CF clear
BL = INT 01h handling (see Table 39-20)
ES:DI buffer filled

**39-20 QEMM "set CPU debug registers" calling convention:**

**Registers at call:**
AX = 1002h
BL = INT 01h handling

## INT 67h Function 3Fh, Subfunction 5145h

00h reflect all debugging exceptions as
V86-mode INT 01h's
else convert debugging exceptions
other than single-step into
V86-mode INT 03h's, single-step
to INT 01h's
ES:DI -> buffer containing debug registers (8 DWORDs)

**Return Registers:**
CF clear

**Details:** This function is identical to INT 67h Function DE09h (Chapter 29) if BL=01h.

The INT 01h handling flag is set to 01h by the general-protection violation handler for certain privileged instructions.

---

**39-21 QEMM "get machine status word" calling convention:**

**Registers at call:**
AX = 1003h

**Return Registers:**
CF clear
EAX = contents of CR0

**See also:** INT 67h Function DE07h (Chapter 29)

---

**39-22 QEMM "allocate a 4K page" calling convention:**

**Registers at call:**
AX = 1004h

**Return Registers:**
CF clear if successful
EDX = linear address of allocated page
CF set on error

**See also:** INT 67h Function DE04h (Chapter 29)

---

**39-23 QEMM "free 4K page" calling convention:**

**Registers at call:**
AX = 1005h free 4K page
EDX = linear address of page to free

**Return Registers:**
CF clear

**See also:** INT 67h Function DE05h (Chapter 29)

---

**39-24 QEMM Function 1006h calling convention:**

**Registers at call:**
AX = 1006h NOP

**Return Registers:**
CF set

---

**39-25 QEMM "get maximum physical memory address" calling convention:**

**Registers at call:**
AX = 1007h

**Return Registers:**
CF clear
EDX = physical address of highest 4K memory page

**See also:** INT 67h Function DE02h (Chapter 29)

---

**39-26 QEMM "get physical address of page in first megabyte" calling convention"**

**Registers at call:**
AX = 1008h
CX = page number (linear address shifted right 12 bits)

**Return Registers:**
CF clear
EDX = linear address of page

**See also:** Function 1F00h (Table 39-76)

---

**39-27 QEMM "switch to protected mode" calling convention:**

**Registers at call:**
AX = 1009h
ESI = linear address in first megabyte
of system register values (see Table 29-2)
interrupts disabled

**Return Registers:**
interrupts disabled
GDTR, IDTR, LDTR, TR loaded
SS:ESP must have at least 16 bytes space, and
the entry point is required to set up a new
stack before enabling interrupts
EAX, ESI, DS, ES, FS, GS destroyed

**See also:** INT 67h Function DE0Ch (Chapter 29)

### 39-28 QEMM "switch back to virtual-86 mode" calling convention:

**Registers at call:**
AX = 100Ah
DS = selector for data segment from Function 1000h (see Table 39-18)
SS:ESP in first megabyte of linear memory
interrupts disabled
STACK: QWORD return address from FAR call to 32-bit segment
DWORD EIP
DWORD CS
DWORD reserved for EFLAGS
DWORD ESP
DWORD SS
DWORD ES
DWORD DS
DWORD FS
DWORD GS

**Return Registers:**
processor switched to virtual86 mode with interrupts disabled, all segment registers loaded, and EAX destroyed

### 39-29 QEMM "get memory type map" calling convention:

**Registers at call:**
AH = 11h
AL = zero/nonzero *unknown* (set by QEMM.COM but apparently ignored by QEMM 6.00)
ES:DI -> 256-byte buffer for memory types

**Return Registers:**
CF clear
BL = *unknown*
ES:DI buffer filled

**Details:** Each byte of the buffer corresponds to a 4K page, and contains the type of that page (see Table 39-30).

### 39-30 Memory page types:

| | |
|---|---|
| 00h | mappable |
| 02h | mapped ROM |
| 03h | high RAM |
| 04h | excluded |
| 05h | video |
| 06h | ROM |
| 07h | adapter ROM |
| 08h | split ROM |
| 09h | page frame |
| 0Ah | RAMmable |
| 0Bh | conventional |

### 39-31 QEMM "get HIRAM chain" calling convention:

**Registers at call:**
AH = 12h

**Return Registers:**
CF clear
BX = segment of first MCB in high memory, or 0000h if no high memory

### 39-32 QEMM "*VIDRAMEGA*" calling convention:

**Registers at call:**
AX = 1300h *VIDRAMEGA*
BL = direction
  00h copy *unknown*
  nonzero copy *unknown* (reverse)

**Return Registers:**
CF clear
AL = status
  00h if all pages clean
  01h if any page dirty

## INT 67h Function 3Fh, Subfunction 5145h

### 39-33 QEMM Function 1301h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 1301h check if *unknown* | CF clear |
| DX:DI = start address of range to check *unknown* | CX = *unknown* (0000h or 1000h) |
| CX = length of range | |

### 39-34 QEMM Function 1302h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 1302h *unknown* | CF clear |
| BL = *unknown* | *others, if any, unknown* |
| BH = *unknown* | |
| CX = *unknown* | |
| SI = offset of *unknown* | |
| DI = offset of *unknown* | |
| *others, if any, unknown* | |

**Details:** This function disables certain interrupts at the two 8259 PICs during execution; it also modifies the CRT controller settings during execution under certain circumstances.

### 39-35 QEMM Function 1303h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 1303h *EMS allocation* | CF clear if successful |
| BX = number of pages (less 1) of EMS to allocate. | DX = EMS handle<br>CF set on error |

### 39-36 QEMM Function 1304h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 1304h EMS deallocation | CF clear |
| DX = EMS handle | |

### 39-37 QEMM Function 1305h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 1305h *unknown* | CF clear |
| CX = *unknown* | |

**Details:** This function disables certain interrupts at the two 8259 PICs during execution (see Table 39-44).

### 39-38 QEMM "set DESQview critical section counter address" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 1306h | CF clear |
| ES:BX -> WORD DESQview critical section counter or 0000h:0000h | |

**Details:** This function also sets a pointer in the low-memory part of QEMM to the current value of INT 15h if ES:BX is not 0000h:0000h.

### 39-39 QEMM Function 1307h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 1307h *unknown* | CF clear |

**Details:** This function disables certain interrupts at the two 8259 PICs during execution (see Table 39-44).

### 39-40 QEMM Function 1308h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 1308h *unknown* | CF clear |
| BL = *unknown* (zero/nonzero) | |

**39-41 QEMM "Hercules mode-change support" calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 1309h | CF clear |
| ES:BX -> new address for Hercules mode-change callback | |

**Details:** The callback function is called whenever the CRTC mode register is written, with AL set to the value written.

**39-42 QEMM "virtualize EGA/VGA I/O ports 03C8h/03C9h" calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 130Ah | CF clear |
| CX:DX -> buffer for *storing CRTC register contents* or 0000h:0000h to disable. | |

**39-43 QEMM Function 130Bh calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 130Bh *unknown* | CF clear |
| BL = *unknown* | *others, if any, unknown* |

**39-44 QEMM "set interrupts to mask" calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 130Ch | CF clear |
| BX = interrupts to mask out during Functions 1302h, 1307h, 1308h, 130Dh, and 1310h (BL = master PIC, BH = slave PIC) | |

**39-45 QEMM Function 130Dh calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 130Dh *unknown* | CF clear |
| *unknown* | |

**Details:** This function disables certain interrupts at the two 8259 PICs during execution (see Table 39-44).

**39-46 QEMM Function 130Eh calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 130Eh *unknown* (modifies CRT controller setup) | CF clear |
| *unknown* | |

**39-47 QEMM Function 130Fh calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 130Fh reset *unknown* | CF clear |

**39-48 QEMM Function 1310h calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 1310h *unknown* | CF clear |
| *others, if any, unknown* | |

**Details:** This function disables certain interrupts at the two 8259 PICs during execution (see Table 39-44).

**39-49 QEMM Function 1311h calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 1311h set *unknown* | CF clear |
| BL = *unknown* | |

**39-50 QEMM Function 1312h calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 1312h (v6.02) *NOP* | none |

**Details:** This function is called by DV 2.42, but appears to be a NOP in QEMM 6.02.

## INT 67h Function 3Fh, Subfunction 5145h

### 39-51 QEMM Function 1400h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 1400h *unknown* | AX = *unknown* |
| ES:DI -> *unknown* data structure (see Table 39-52) | |
| BL = *unknown* | |

### 39-52 Format of data structure for Function 1400h:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | *unknown* |
| 02h | DWORD | far pointer to *unknown* |
| 06h | DWORD | far pointer to *unknown* pointer array (see Table 39-53) |
| 0Ah | DWORD | far pointer to *unknown* |
| 0Eh | DWORD | *unknown* |
| 12h | WORD | segment of *unknown* |
| 14h | DWORD | far pointer to *unknown* |
| | | *possibly additional fields* |

### 39-53 Format of pointer array:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of pointers to follow |
| 02h | N DWORDs | far pointers to *unknown* |

**Details:** QEMM converts the pointers into linear addresses in place.

### 39-54 QEMM Function 1401h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 1401h *unknown* | CF clear |
| | *unknown* |

### 39-55 QEMM Function 1402h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 1402h *unknown* | CF clear |
| BL = function<br>00h NOP<br>01h *unknown*<br>02h *unknown*<br>other *unknown* | *others, if any, unknown* |
| ES:DI -> *unknown* (see Table 39-56) | |

### 39-56 Format of data structure for Function 1402h:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of *unknown* (X, word at X:0136h set to X) |
| 02h | WORD | segment of *unknown* (word at X:0124h set to this) |
| 04h | WORD | number of paragraphs of *unknown* |
| 06h | 3 WORDs | *unknown* (copied to X:0000h) |
| 0Ch | WORD | *unknown* |

### 39-57 QEMM Function 1403h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 1403h add *unknown* to list and *unknown* (execute func 1406h) | CF clear |
| ES:DI -> *unknown* structure added to end of *unknown* list (at least 31 bytes, DWORD at offset 06h used for storing pointer to next struc, WORD at offset 00h seems to be a key or index) | |

**39-58 QEMM Function 1404h calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 1404h NOP | none |

**39-59 QEMM Function 1405h calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 1405h remove *unknown* from *unknown* list | CF clear |
| BX = *key* | |

**39-60 QEMM Function 1406h calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 1406h *unknown* | CF clear |
| *unknown* | *unknown* |

**39-61 QEMM Functions 1407h to 140Ah calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 1407h to 140Ah | CF clear |
| *others, if any, unknown* | *others, if any, unknown* |

**39-62 QEMM Function 140Bh calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 140Bh *unknown* | CF clear |
| BX = *unknown* | SI = *segment of 256-byte buffer* |

**39-63 QEMM Function 15h calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AH = 15h *unknown* | CF clear |
| ES:BX -> *unknown* or 0000h:0000h | |

**39-64 QEMM v5.00+ "get memory access status" calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 1600h | ES:DI buffer filled |
| ES:DI -> 256-byte buffer for access status info | |

**Details:** Each byte of the buffer indicates the status of a 4K page (bit 0 set if read, bit 1 set if written).

**39-65 QEMM v5.00+ "set memory access status" calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = 1601h | CF clear |
| ES:DI -> 256-byte buffer containing access statuses (see Table 39-64). | |

**39-66 QEMM v5.00+ "get memory usage statistics" calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AH = 17h get memory usage statistics | CF clear |
| ES:DI -> 81-byte buffer for memory statistics (see Table 39-83) | |

**39-67 QEMM v5.11+ "check if buffer is conventional memory" calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AH = 18h | CF clear |
| ES:BX = starting address | AL = status<br>00h one or more pages is remapped<br>01h all pages in range are conventional memory (physical address == virtual address) |
| CX = number of 4K pages | |

## INT 67h Function 3Fh, Subfunction 5145h

### 39-68 QEMM v5.11+ Function 19h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 19h NOP | CF set |

### 39-69 QEMM v5.11+ "I/O port access" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 1Ah | CF clear |
| DX = port number | |
| AL = subfunction | |
| 00h get byte from I/O port | BL = port value |
| 01h send byte to I/O port | |
| BL = value to send | |
| 02h send byte to I/O port, get byte from following port | BL = value read |
| BH = value to send | |
| 03h send bytes to two consecutive I/O ports | |
| BH = value for first I/O port (DX) | |
| BL = value for second I/O port (DX+1) | |

### 39-70 QEMM v5.11+ "MS Windows 3.x support" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 1Bh | |
| AL = subfunction | |
| 00h get EMM Import Structure address | CF set on error |
| ES:DI -> buffer for EMM import data structure (see Table 39-71) | CF clear if successful |
| **See also:** INT 21h Function 4402h | CF set on error |
| Subfunction 01h (Chapter 42) | CF clear if successful |
| 01h *unknown* | |
| | CF set on error |
| | CF clear if successful |
| 02h *unknown* | |
| | *unknown* |
| 03h MS Windows initializing | |
| CX = *segment from which Windows init broadcast made* | |
| DL = *Windows startup flags* | |
| DI = Windows version number (major in upper byte) | CF clear |
| **See also:** INT 2Fh Function 1605h (Chapter 43) | |
| 04h MS Windows terminating | CF clear |
| | AL = type |
| 05h *determine whether program is driver* | 01h if string ends in ".DRV" |
| DS:DX -> ASCIZ filename | FFh if string ends in "GDI.EXE" |
| | 00h otherwise |
| | CF clear |
| 06h *unknown* | |
| CX = length of data pointed at by DS:DX | |
| DS:DX -> *unknown* | |
| 07h **BUG:** QEMM 6.00-7.01 accept this and branch randomly | CF set |
| else | |

### 39-71 Format of EMM Import data structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | physical address of EMM import structure |
| 04h | BYTE | major version (v6.00 sets to 01h) |
| 05h | BYTE | minor version (v6.00 sets to 00h/0Bh) |

### 39-72 QEMM v5.11+ "protected-mode hardware interrupt handlers" calling convention:

**Registers at call:**

AH = 1Ch
AL = subfunction
  00h *restore* IRQ0-7 handlers
  01h *set* IRQ0-7 handlers
    ES:DI -> 8 DWORDScontaining *unknown*
  02h *restore* IRQ8-15 handlers
  03h *set* IRQ8-15 handlers
    ES:DI -> 8 DWORDScontaining *unknown*
 #BReturn Registers:
 CF clear

**BUG:** although the jump table only contains four entries, QEMM 6.00 will attempt to use it for any value of AL between 00h and 2Ah, thus branching unpredictably for AL=04h-2Ah; QEMM v7.01 behaves similarly for AL=04h-1Bh.

### 39-73 QEMM v6.0x only "Stealth interrupts" calling convention

**Registers at call:**

AH = 1Dh
AL = subfunction
  00h switch to pre-Stealth interrupt vector table
  01h restore user interrupt vector table

**Return Registers:**

CF clear if AL=00h or 01h
CF set if AL=other on entry

**Details:** Subfunction 00h also switches the VGA Save table pointer (0040h:00A8h) and overwrites the vectors currently assigned for use by the two interrupt controllers (see INT 67h Function DE0Ah in Chapter 29) with the vectors for INT 08h-0Fh and 70h-77h (to avoid crashing the system).

Interrupts should be disabled around the Function 1D00h and 1D01h calls because QEMM does not modify the memory maps to map in ROM, so an interrupt could be disastrous clears any pending IRQ7 at end of function else.

Functions 1Dxxh are not supported by QEMM v7.01, and always return CF set.

### 39-74 QEMM v6.00+ "Stealth information" (documented) calling convention:

**Registers at call:**

AH = 1Eh
AL = subfunction
  00h "QEMM_GET_INFO"
    get Stealth configuration

  01h
"QEMM_GET_STEALTH_COUNT"

| | Return Registers: |
|---|---|
| get number of Stealth'ed ROMs | |
| 02h "QEMM_GET_STEALTH_LIST" get Stealth'ed ROM info<br>ES:DI -> buffer for Stealth ROM info (see Table 39-84) | BL = memory flags (see Table 39-75)<br>BH = reserved (always 00h for v6.00)<br>CL = stealth type (00h none,46h Frame,4Dh Map)<br>CH = suspend/resume interrupt (00h none)<br>DX = reserved (always 0000h for v6.00)<br>SI = reserved (always 0000h for v6.00)<br>DI = reserved (always 0000h for v6.00) |
| else | CF clear<br>BX = number of Stealth'ed ROMs |
| | CF clear<br>BX = number of Stealth'ed ROMs<br>ES:DI buffer filled |
| | CF set |

### 39-75 Bitfields for memory flags:

| | |
|---|---|
| bit 0 | conventional memory sorted |
| bit 1 | conventional memory filled |
| bit 2 | *unknown* |
| bit 3 | *unknown* |
| bit 4 | expanded memory is in use |
| bit 5 | *unknown* |

### 39-76 QEMM v6.00+ "page table manipulation" (documented) calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 1Fh | |
| AL = subfunction | |
| 00h "QEMM_GET_PTE" get page table entry<br>CX = page number | CF clear<br>EDX = page table entry |
| 01h "QEMM_SET_PTE" set page table entry<br>CX = page number<br>EDX = new page table entry<br>**See also:** Table 39-26 | CF clear |
| else | CF set |

### 39-77 QEMM v6.00+ "asynchronous disk access support" (documented) calling convention:

**Registers at call:**

AH = 20h

AL = subfunction

00h "QEMM_GET_VHI_INFO" get VirtualHDIRQ information

| | Return Registers: |
|---|---|
| 01h "QEMM_SET_VHI_FINO"<br>set VirtualHDIRQ state<br>BL bit 0 = new VirtualHDIRQ state<br>else | CF clear<br>BL = flags<br>bit 7: VirtualHDIRQ setting respected (set if Stealth active)<br>bits 6-1 reserved<br>bit 0: VirtualHDIRQ currently enabled (INT 15h Function 90h suppressed when enabled) |
| | CF clear<br>BL = old VHI setting (bits 0 and 7, see above) |
| | CF set |

**39-78 QEMM v6.00+ "Stealth support" (documented) calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AH = 21h | |
| AL = subfunction | |
| 00h<br>"QEMM_COPY_STEALTH_ROMS"<br>copy data from Stealthed addresses<br>DS:SI -> start address of hidden memory to copy<br>ES:DI -> buffer for copied data<br>ECX = number of bytes to copy | CF clear if successful<br>CF set on error (DS:SI < C000h:0000h or DS:SI + ECX > 1M) |
| else | CF set |

**39-79 QEMM v6.03+ "DESQview/X support" calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AH = 22h DESQview/X support | |
| AL = subfunction | |
| 00h get *unknown* | CF clear<br>ES:DI -> *unknown* |
| 01h set *unknown*<br>ES:DI -> *unknown* or 0000h:0000h | CF clear if successful<br>CF set on error |

**39-80 QEMM v6.04+ Function 23h calling convention:**

Registers at call:

AH = 23h *unknown*

AL = subfunction

00h get *unknown*

BX = which *unknown* to get (must be 0000h for v6.04)

01h set *unknown*

BX = which *unknown* to set (must be 0000h for v6.04)

ES:DI -> *unknown*

02h clear specified *unknown*

## INT 67h Function 3Fh, Subfunction 5145h

| | **Return Registers:** |
|---|---|
| BX = which *unknown* to clear<br>(must be 0000h for v6.04) | |
| FFh clear all *unknown* | CF clear if successful<br>ES:DI -> *unknown*<br>CF set on error |
| else | CF clear if successful<br>CF set on error |
| | CF clear if successful<br>CF set on error |
| | CF set |

### 39-81 QEMM v7.01 "ST-DBL support" calling convention:

**Registers at call:**

AH = 24h ST-DBL support
AL = subfunction
  00h set *unknown*
    EDX -> information table
    (EDX = segment SHL 16 + offset)
  01h *unknown*

### 39-82 QEMM unsupported functions calling convention:

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = other than listed above | CF set |

### 39-83 Format of QEMM 6.0 memory statistics:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | 01h if Shadow RAM found, 00h otherwise |
| 01h | DWORD | initial conventional memory in bytes |
| 05h | DWORD | initial extended memory in bytes |
| 09h | DWORD | initial expanded memory in bytes |
| 0Dh | DWORD | initial "top" or "shadow" memory in bytes |
| 11h | DWORD | Unavailable conventional memory in bytes |
| 15h | DWORD | Unavailable extended memory in bytes |
| 19h | DWORD | Unavailable expanded memory in bytes |
| 1Dh | DWORD | Unavailable "top" or "shadow" memory in bytes<br>Add to offset 49h for Total unavailable top/shadow memory |
| 21h | DWORD | QEMM code size in bytes |
| 25h | DWORD | QEMM data size in bytes |
| 29h | DWORD | bytes used for TASKS= |
| 2Dh | DWORD | DMA buffer size |
| 31h | DWORD | bytes used for MAPS= |
| 35h | DWORD | bytes of high RAM |
| 39h | DWORD | bytes used by mapped ROMs |
| 3Dh | DWORD | bytes of conventional memory provided by QEMM |
| 41h | DWORD | bytes of extended memory NOT converted by QEMM (EXT=xxx) |
| 45h | DWORD | bytes of EMS/XMS pool memory provided by QEMM |

| | | |
|---|---|---|
| 49h | DWORD | Unavailable "top" or "shadow" memory in bytes<br>Add to offset 1Dh for Total unavailable top/shadow memory |
| 4Dh | DWORD | conventional memory overhead in bytes (set to 0 by QEMM.COM prior to call) |

**39-84 Format of Stealth ROM information [array]:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | starting segment of ROM |
| 02h | WORD | length of ROM in paragraphs |

---

**INT FFh** *QEMM-386.SYS v6.0+*
***internal*** *(undocumented)*

**Details:** This interrupt requires that a byte in the conventional-memory stub be set to the desired function number (00h through 0Ch).

## CHAPTER 40

# Qualitas 386MAX

386MAX, from Qualitas Inc., Bethesda, MD, was the first 386 memory manager on the market, and set the standard for this entire class of products. A companion product, BlueMAX, is sold for Micro Channel machines. Both products provide all memory-management basics, and have supported EMS, extended memory, VDS, and the virtual control program interface (VCPI) for years. This chapter is devoted to the API of these Qualitas programs.

---

### INT 21h Function 4402h
### *IOCTL INPUT - GET STATE*

*386MAX v6.00+*
*(undocumented)*

**Registers at call:**
AX = 4402h
BX = file handle for device "386MAX$$"
CX = number of bytes to read
DS:DX -> BYTE 03h followed by 386MAX state buffer (see Table 40-1)

**Return Registers:**
CF clear if successful
    buffer at DS:DX+1 filled
    AX = number of bytes actually copied
CF set on error
    AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Details:** If the value given in CX is less than the size of the state record (5Ah for v6.01, 66h for v7.00), only a partial state record will be returned.

The state is 40h bytes for 386MAX (actually ASTEMM) v2.20 ("386MAX$$" did not exist yet, use "QMMXXXX0" and then "EMMXXXX0" instead) and 56h bytes for v5.11.

To invoke 386MAX API functions, push DX onto the stack, load DX with the word at offset 25h in the returned state, load all other registers as needed for the desired function, and execute an OUT DX,AL or OUT DX,AX; DX will be set to the pushed value on return if it is not otherwise modified by the API function. For safety, in case a function is not supported or 386MAX is not present, SP should be saved and restored around the API call.

Windows 3.1 Standard mode, LAN Manager, and Windows for Workgroups all use the 386MAX API; LAN Manager and Windows for Workgroups reportedly make some calls incorrectly.

**See also:** Function 4403h Subfunction 03h, QEMM INT 67h Function 3Fh (Chapter 39)

**40-1 Format of 386MAX v6.01+ state:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -1 | BYTE | (call) 03h |
| 00h | 6 BYTEs | signature "386MAX" |
| 06h | 4 BYTEs | version string "N;NN" (i.e. "6;01" for v6.01) |
| 0Ah | WORD | segment of low-memory portion of 386MAX.SYS |
| 0Ch | 2 BYTEs | *unknown* |
| 0Eh | WORD | segment of *unknown* memory block or 0000h |

| | | |
|---|---|---|
| 10h | WORD | bit flags 1 (see Table 40-2) |
| 12h | WORD | (v6.01) starting address of video memory in KB<br>(v7.00) low memory size in bytes |
| 14h | 2 BYTEs | *unknown* |
| 16h | WORD | total high DOS memory in KB |
| 18h | 2 BYTEs | *unknown* |
| 1Ah | WORD | available shared memory in KB |
| 1Ch | WORD | KBytes extended memory used by 386MAX |
| 1Eh | 2 BYTEs | *unknown* |
| 20h | WORD | total extended memory in KB |
| 22h | WORD | IO port to write (OUT DX,AL) to invoke 386MAX INT 15h functions |
| 24h | WORD | IO port to write (OUT DX,AL) to invoke 386MAX API functions |
| 26h | WORD | *unknown* (depends on DOS version) |
| 28h | 2 BYTEs | *unknown* |
| 2Ah | DWORD | machine type (see Table 40-10) |
| 2Eh | WORD | *unknown* |
| 30h | WORD | *unknown* |
| 32h | WORD | system configuration flags (see Table 40-3) |
| 34h | WORD | debugging flags 1 (see Table 40-4) |
| 36h | WORD | debugging flags 2 (see Table 40-5) |
| 38h | 2 BYTEs | *unknown* |
| 3Ah | WORD | *segment of first MCB in high memory chain* |
| 3Ch | WORD | feature flags 1 (see Table 40-7) |
| 3Eh | WORD | feature flags 2 (see Table 40-8) |
| 40h | WORD | feature flags 3 (see Table 40-9) |
| 42h | WORD | *segment of first 386MAX control block* (see Table 40-11) |
| 44h | WORD | amount of memory to report available on INT 15h Function 88h |
| 46h | 4 BYTEs | *unknown* |
| 4Ah | WORD | number of K at start of address space swapped with fast extended memory (SWAP= parameter) |
| 4Ch | 2 BYTEs | *unknown* |
| 4Eh | WORD | *unknown* |
| 50h | WORD | debugging flags 3 (see Table 40-6) |
| 52h | DWORD | old INT 21h |
| 56h | DWORD | pointer to 386MAX's EMS (INT 67h) handler |
| ---386MAX v7.00--- | | |
| 5Ah | DWORD | KB of extended memory managed by 386MAX |
| 5Eh | DWORD | bytes of extended memory (EXT= parameter) |
| 62h | 4 BYTEs | *unknown* |

**40-2 Bitfields for bit flags 1:**

| | |
|---|---|
| bit 1 | *unknown* |
| bit 2 | *unknown* |
| bit 3 | *unknown* (cleared by calling INT 67h functions) |
| bit 4 | *high RAM present* |
| bit 5 | 386MAX in AUTO mode |
| bit 6 | 386MAX enabled |
| bit 7 | 386MAX is providing EMS services |
| bit 8 | *unknown* |
| bit 9 | *A20 enabled* (see INT 15h Function 2402h in Chapter 4) |

## INT 21h Function 4402h

| | |
|---|---|
| bit 10 | Weitek support enabled |
| bit 11 | *unknown* |
| bit 12 | *unknown* |
| bit 13 | QPMS has been used |
| bit 14 | *unknown* |
| bit 15 | *unknown* |

**40-3 Bitfields for system configuration flags:**

| | |
|---|---|
| bit 1 | *ROM compressed* |
| bit 3 | *unknown* |
| bit 5 | 386MAX loaded into high memory |
| bit 6 | Microchannel bus |
| bit 7 | Weitek math coprocessor detected |
| bit 11 | PC/XT (thus only single 8259 interrupt controller present, DMA only in 1st megabyte, etc.) |
| bit 13 | LMLTOP= specified |
| bit 15 | *unknown* |

**40-4 Bitfields for debugging flags 1:**

| | |
|---|---|
| bit 0 | DEBUG=LED |
| bit 1 | DEBUG=X67 |
| bit 2 | DEBUG=INV |
| bit 3 | DEBUG=EMSPTED |
| bit 4 | DEBUG=JMP |
| bit 5 | DEBUG=CALL |
| bit 6 | DEBUG=HLT |
| bit 7 | DEBUG=PMR |
| bit 8 | DEBUG=CR3 |
| bit 9 | DEBUG=CAPS or DEBUG=INT |
| bit 10 | DEBUG=RC |
| bit 11 | DEBUG=ROM |
| bit 12 | DEBUG=XM |
| bit 13 | DEBUG=SOR |
| bit 14 | DEBUG=XR |
| bit 15 | DEBUG=EMSERR |

**40-5 Bitfields for debugging flags 2:**

| | |
|---|---|
| bit 0 | DEBUG=ROMSWAP |
| bit 1 | DEBUG=UNSHADOWROM |
| bit 2 | DEBUG=COMPROM |
| bit 3 | DEBUG=DPMIPHYS |
| bit 4 | DEBUG=ALLROM |
| bit 5 | DEBUG=VMS |
| bit 6 | DEBUG=XMS |
| bit 7 | DEBUG=I06 |
| bit 8 | DEBUG=VCPI |
| bit 9 | DEBUG=XDMA |
| bit 10 | DEBUG=X09 |
| bit 13 | DEBUG=I67 |
| bit 14 | DEBUG=EVM |
| bit 15 | DEBUG=EMSSAVE or DEBUG=VDS |

### 40-6 Bitfields for debugging flags 3:

| | |
|---|---|
| bit 10 | DEBUG=EPM |
| bit 12 | DEBUG=ABIOS |
| bit 13 | DEBUG=XMSPTED |
| bit 14 | DEBUG=TIME |
| bit 15 | DEBUG=SCRUB |

### 40-7 Bitfields for feature flags 1:

| | |
|---|---|
| bit 1 | Weitek present |
| bit 2 | no DPMI services |
| bit 3 | NODMA |
| bit 4 | TERSE |
| bit 5 | NOROM |
| bit 6 | NOPARITY |
| bit 8 | NOFLEX (IGNOREFLEXFRAME) |
| bit 11 | don't create UMBs |
| bit 12 | don't backfill below video memory (NOLOW) |
| bit 13 | FRAME= specified |
| bit 14 | EXT= specified |
| bit 15 | NOEMS, allow prior expanded memory manager to provide EMS |

### 40-8 Bitfields for feature flags 2:

| | |
|---|---|
| bit 0 | UNSHIFT specified (FORCEA20 disabled) |
| bit 1 | NOXRAM |
| bit 2 | NOSCSI specified |
| bit 3 | SCREEN specified |
| bit 4 | enabled EISADMA |
| bit 5 | slow DMA |
| bit 6 | RESETKEYB specified |
| bit 7 | *unknown* |
| bit 9 | TOP384 |
| bit 11 | NOWARMBOOT |
| bit 12 | USE= specified |
| bit 13 | ROM= specified |

### 40-9 Bitfields for feature flags 3:

| | |
|---|---|
| bit 0 | Windows3 support enabled |
| bit 1 | SHADOWROM |
| bit 2 | don't compress ROM (NOCOMPROM) |
| bit 3 | *unknown* (related to PRGREG=) |
| bit 4 | *unknown* (related to PRGREG=) |
| bit 5 | SHADOWRAM |
| bit 6 | DOS4 specified |
| bit 7 | NOLOADHIGH |
| bit 8 | NOPULSE |
| bit 11 | FORCEA20 |
| bit 12 | DMA buffer enabled |
| bit 13 | NOSCRUB |
| bit 15 | NOFRAME |

## INT 21h Function 4402h

### 40-10 Bitfields for machine type:

| | |
|---|---|
| bit 12 | Amstrad |
| bit 13 | Epson |
| bit 14 | Zenith Data Systems |
| bit 15 | "ASEM" |
| bit 16 | NEC |
| bit 17 | "HPRS" model codes 69h and 6Ah |
| bit 18 | Dell |
| bit 19 | "CA" |
| bit 20 | ITT (Xtra Business Systems/Alcatel) |
| bit 21 | Toshiba 5100 |
| bit 22 | Olivetti |
| bit 23 | Quadram Quad386 (BIOS model FEh, submodel A6h) |
| bit 24 | *Tandy* |
| bit 25 | AST 386 |
| bit 26 | INBOARD, *unknown* version |
| bit 27 | INBOARD, *unknown* version |
| bit 28 | INBOARD, *unknown* version |
| bit 29 | "HPRS" |
| bit 30 | Compaq 386 |
| bit 31 | JET386 |

### 40-11 Format of 386MAX control block:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of next block (FFFFh if last) |
| 02h | WORD | segment of previous block (FFFFh if first) |
| 04h | 12 BYTEs | filename |
| 10h | WORD | resident size in paragraphs |
| 12h | WORD | *environment size* |
| 14h | WORD | real prsent environment size + 1 (0000h if ENVSAVE used) |
| 16h | 2 BYTEs | *unknown* |
| 18h | DWORD | initial size or SIZE=n in 386LOAD commandline |
| 1Ch | DWORD | *SIZE=-1* |
| 20h | DWORD | *SIZE=* |
| 24h | BYTE | PRGREG= if specified, else FFh |
| 25h | BYTE | ENVREG= if specified, else FFh |
| 26h | BYTE | FlexFrame (00h not present, 01h present) |
| 27h | 3 BYTEs | *unknown* |
| 2Ah | BYTE | GROUP= or 00h if not present |
| 2Bh | BYTE | *unknown* |
| 2Ch | WORD | PSP |

### 40-12 Format of high memory info record:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment address of memory region |
| 02h | WORD | size of memory region in paragraphs |
| 04h | BYTE | *type or flags*<br>00h if locked out<br>02h if EMS page frame<br>04h if high RAM<br>42h if ROM |
| 05h | BYTE | *unknown* |

### 40-13 Format of ROM shadowing record:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | *segment of ROM* |
| 02h | WORD | *s* *egment of ROM* |
| 04h | 2 BYTEs | *unknown* |
| 06h | WORD | size of shadowed ROM in paragraphs |
| 08h | 4 BYTEs | *unknown* |

### 40-14 Values for memory type:

| | |
|---|---|
| 00h | unused by EMS |
| 01h | DOS |
| 04h | *page frame overlapping ROM* |
| 80h | high memory |
| 84h | *page frame* |
| 87h | *video ROM* |

**Note:** The type may be 00h (unused) if the 16K page is split among different uses (such as ROM and high RAM).

### 40-15 386MAX "get high memory information" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| STACK: WORD value for DX | AH = status (see Table 38-1) |
| AH = 01h | CX = number of records placed in buffer |
| ES:DI -> buffer for array of high memory info records (see Table 40-12) | DX popped from stack |

### 40-16 386MAX "*get shadowed ROM info*" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 02h | AH = status (see Table 38-1) |
| ES:DI -> buffer for array of ROM shadowing records (see Table 40-13) | CX = number of records placed in buffer<br>DX popped from stack |

### 40-17 386MAX "get 386MAX state" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 03h | AH = 00h (successful) |
| ES:DI -> 90-byte buffer for state (see Table 40-1) | buffer filled<br>DX popped from stack |

**Details:** Unlike 386MAX INT 21h Function 4402h, this function omits the first byte specifying the state buffer version.

### 40-18 386MAX "get memory types" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 04h | AH = status (see Table 38-1) |
| ES:DI -> buffer for memory type info (array of bytes, one per 16K page) (see Table 40-14) | CX = number of bytes placed in buffer<br>DX popped from stack |

### 40-19 386MAX Function 05h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 05h get *unknown* | AH = status (see Table 38-1) |
| AL = A20 control (00h enable A20 first, 01h leave unchanged) | CX = number of 4K pages reported on<br>DX popped from stack |
| CX = number of 4K pages to report (0000h for default) | *others, if any, unknown* |
| SI = first K to report (rounded down to 4K page) | |
| ES:DI -> buffer for returned info on *unknown* | |

## INT 21h Function 4402h

### 40-20 386MAX "get memory speed info" calling convention:

**Registers at call:**
AH = 06h
ES:DI -> buffer for memory speed records (see Table 40-57)

**Return Registers:**
AH = 00h (successful)
CX = number of bytes placed in buffer
DX popped from stack

**Details:** This function can take over a second to execute.

### 40-21 386MAX Function 07h calling convention:

**Registers at call:**
AH = 07h *unknown*
DX = EMS handle (on stack)
*others, if any, unknown*

**Return Registers:**
AH = status (see Table 38-1)
DX popped from stack
*others, if any, unknown*

### 40-22 386MAX "EMM2_GOREAL" calling convention:

**Registers at call:**
AH = 08h check whether possible to disable 386MAX
AL = *unknown* (00h or nonzero)

**Return Registers:**
AH = status (00h OK, A4h not possible at this time)
DX popped from stack

**Details:** If AL=00h on call, this function always returns success.

### 40-23 386MAX Function 09h calling convention:

**Registers at call:**
AH = 09h toggle *unknown* flags
BX = bitmask of bits of *unknown* flags (word at state+10h) to toggle

**Return Registers:**
AH = 00h (successful)
DX popped from stack

**Details:** This function enables A20 first.

### 40-24 386MAX Function 0Ah calling convention:

**Registers at call:**
AH = 0Ah toggle *unknown* flags
BX = bitmask of bits of *unknown* *(word at state+32h) to toggle*

**Return Registers:**
AH = 00h (successful)
DX popped from stack

**Details:** This function enables A20 first.
Performs *unknown* additional actions if bit 3 is set after the specified bits are toggled.

### 40-25 386MAX Function 0Bh calling convention:

**Registers at call:**
AH = 0Bh toggle *unknown* flags
BX = bitmask of bits of *unknown* *(word at state+34h) to toggle*

**Return Registers:**
AH = 00h (successful)
DX popped from stack

**Details:** This function enables A20 first.

### 40-26 386MAX Function 0Ch calling convention:

**Registers at call:**
AH = 0Ch toggle *unknown* flags
BX = bitmask of bits of *unknown* *(word at state+40h) to toggle*

**Return Registers:**
AH = 00h (successful)
DX popped from stack

**Details:** This function enables A20 first.

### 40-27 386MAX "specify 386MAX high-memory location" calling convention:

**Registers at call:**
AH = 0Dh
BX = segment address of high-memory real-mode portion of 386MAX
CX = *current segment of real-mode stub*

**Return Registers:**
AH = status (00h successful)
DX popped from stack
*others, if any, unknown*

**40-28 386MAX "CRT controller register virtualization" calling convention:**

**Registers at call:**
AH = 0Eh
AL = subfunction
    00h allow access to CRTC I/O ports
      03B4h/03B5h, 03D4h/03D5h
    01h trap accesses to CRTC I/O ports

**Return Registers:**
AH = status (see Table 38-1)
DX popped from stack

**40-29 386MAX "reboot system" calling convention:**

**Registers at call:**
AH = 0Fh

**Return Registers:**
never returns

**40-30 386MAX "get high memory information" calling convention:**

**Registers at call:**
AH = 11h
ES:DI -> 96-byte buffer for high memory info

**Return Registers:**
AH = 00h (successful)
ES:DI buffer filled
DX popped from stack

**Details:** Each byte in buffer contains bit flags for a 4K page in the A000h-FFFFh region (see Table 40-58).

This function can take over a second to execute, because it also tests the memory.

**40-31 386MAX Function 12h calling convention:**

**Registers at call:**
AH = 12h *unknown*
AL = subfunction
    00h *unknown*
    01h *unknown*
*others, if any, unknown*

**Return Registers:**
AH = 00h (successful) if AL=00h or 01h
AH = 8Fh otherwise
DX popped from stack

**40-32 386MAX "*page protection*" calling convention:**

**Registers at call:**
AH = 13h
AL = subfunction
    00h set all *unknown* 4K pages to
      *read-only*
    01h set all *unknown* 4K pages to *read-write*
*others, if any, unknown*

**Return Registers:**
AH = 00h (successful) if AL=00h or 01h
AH = 8Fh otherwise
DX popped from stack

**40-33 386MAX Function 14h calling convention:**

**Registers at call:**
AH = 14h *unknown*
ES:DI -> 54-byte buffer for *unknown*

**Return Registers:**
AH = status (00h,A4h) (see Table 38-1)
DX popped from stack

**40-34 386MAX Function 15h calling convention:**

**Registers at call:**
AH = 15h *unknown*
*others, if any, unknown*

**Return Registers:**
AH = status (see Table 38-1)
DX popped from stack
*others, if any, unknown*

**40-35 386MAX "get 386MAX memory usage screen" calling convention:**

**Registers at call:**
AH = 16h
ES:DI -> buffer for memory info display
CX = size of buffer in bytes

**Return Registers:**
AH = status (see Table 38-1)
ES:DI buffer filled with '$'-terminated string
  (if large enough to hold entire usage screen)
DX popped from stack

**Details:** The screen is 0303h bytes in v7.00.

### 40-36 386MAX "Windows 3 startup/termination" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 17h | AH = function status (see Table 38-1) |
| AL = subfunction | DX popped from stack |
| 00h Windows3 initializing | CX = Windows status |
| DX (on stack) = Windows startup flags | 0000h if OK for Windows to load<br>nonzero if Windows should not load |
| DI = Windows version number (major in upper byte) | ES:BX -> startup info structure<br>DS:SI -> Virtual86 mode enable/disable callback |
| ES:BX = 0000h:0000h | |
| DS:SI = 0000h:0000h. | *unknown* |
| 01h Windows3 terminating | |
| ES:BX -> *unknown* | |
| DX (on stack) = Windows exit flags | |

### 40-37 386MAX "QPMS (Qualitas Protected Memory Services)" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 18h | AH = status (00h,8Ah,8Bh,8Fh) |
| AL = subfunction | DX popped from stack |
| 00h get QPMS configuration | BX = starting segment of QPMS memory window<br>CX = *number of 4K pages reserved for QPMS*<br>DX = *number of 4K pages in QPMS window* |
| 01h *map QPMS memory page* | |
| BX = *4K page number within memory reserved for QPMS* | |
| CL = *4K page number within QPMS memory window* | |
| 02h mark all QPMS memory read-only | |
| 03h mark all QPMS memory read-write | |

### 40-38 386MAX "get linear address for physical address" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 19h | AH = status (see also Table 38-1) |
| EDX = physical address (low word on stack) | 00h successful<br>EDX = linear address at which physical address may be accessed<br>8Bh physical address currently not addressable |
| | STACK: top word popped |

**Details:** This function enables A20 first.

### 40-39 386MAX "*set page table entry*" calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 1Ah | AH = status (00h,8Bh) (see Table 38-1) |
| EDX = *new page table entry* (low word on stack) | DX popped from stack |
| ESI = *linear address of page* | |

**Details:** This function enables A20 first.

### 40-40 386MAX Function 1Bh calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 1Bh get *unknown* | AH = status |
| | BX = *unknown* |
| | CX = *unknown* |

EDX = physical address of *unknown*
STACK: top word popped

### 40-41 386MAX "get original interrupt vector" calling convention:

**Registers at call:**
AH = 1Ch
AL = interrupt vector (00h-7Fh)

**Return Registers:**
AH = 00h (successful)
EDX = original vector before 386MAX loaded
(segment in high word, offset in low word)
STACK: top word popped

**Details:** No range checking is performed; requests for INTs 80h-FFh will return random values.

### 40-42 386MAX Function 1Dh calling convention:

**Registers at call:**
AH = 1Dh *display string*
SI = *unknown*

**Return Registers:**
AH = 00h (successful)
DX popped from stack
*others, if any, unknown*

### 40-43 386MAX "get memory info" calling convention:

**Registers at call:**
AH = 1Eh
ES:DI -> memory info (see Table 40-59)

**Return Registers:**
AH = status (see Table 38-1)
DX popped from stack
*others, if any, unknown*

### 40-44 386MAX "get DPMI host information" calling convention:

**Registers at call:**
AH = 1Fh

**Return Registers:**
AX = 0000h if successful
BX = DPMI flags (see INT 31h Function 0400h)
CL = CPU type (02h = 80286, 03h = 80386, etc.)
DX = DPMI version supported (DH=major, DL=2-digit minor)
SI = *unknown*
*ES*:DI -> *unknown*
STACK: top word popped

**Details:** This function is a NOP if the NODPMI switch was specified.

### 40-45 386MAX v7.00 Function 20h calling convention:

**Registers at call:**
AH = 20h
AL = *unknown*

**Return Registers:**
AH = status (see Table 38-1)
EDX = *unknown* for *unknown* AL
STACK: top word popped

### 40-46 386MAX v7.00 "STACKS support" calling convention:

**Registers at call:**
AH = 21h
AL = subfunction

00h get STACKS parameters

01h set *unknown* "EMM2_DSTKS"
EBX = *unknown*
ECX = *unknown*
DS = *unknown*
02h *unknown*

| | |
|---|---|
| | **Return Registers:** |
| BL = *unknown* (00h-03h) | AH = status (see Table 38-1) |
| *others, if any, unknown* | DX popped from stack |
| | BX = *unknown* |
| | CX = number of stacks for hardware interrupts |
| | DX = size of each stack in bytes |
| | SI = *unknown* |
| | DI = *unknown* |

### 40-47 386MAX v7.00 Function 22h calling convention:

| Registers at call: | Return Registers |
|---|---|
| AH = 22h *unknown* | |
| AL = 00h *unknown* | |
| AL = nonzero *unknown* | |
| *others, if any, unknown* | |
| AH = status (see Table 38-1) | |
| DX popped from stack | |
| *others, if any, unknown* | |

### 40-48 386MAX v7.00 Function 23h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 23h *unknown* | AH = status (see Table 38-1) |
| AL = subfunction | DX popped from stack |
| 00h set *unknown* | |
| *others, if any, unknown* | |
| 01h *unknown* | |
| *others, if any, unknown* | |
| 02h get *unknown* | CX = number of bytes actually returned |
| CX = size of buffer | |
| ES:DI -> buffer for *unknown* | |
| 03h set *unknown* | |
| ES:DI -> buffer containing ASCIZ *unknown* | |
| 04h get *unknown* | |
| ES:DI -> buffer for ASCIZ *unknown* | |

### 40-49 386MAX v7.00 Function 24h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 24h *unknown* | AH = status (see Table 38-1) |
| AL = subfunction | DX popped from stack |
| 00h get *unknown* | BX = current state of *unknown* (0/1) |
| 01h set *unknown* | |
| BX = new state of *unknown* (0/1) | |
| other *unknown* | |

### 40-50 386MAX v7.00 Function 25h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 25h *unknown* | AH = status (see Table 38-1) |
| *others, if any, unknown* | DX popped from stack |
| | *others, if any, unknown* |

### 40-51 386MAX v7.00 Function 26h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 26h *unknown* | AH = status (see Table 38-1) |
| *others, if any, unknown* | BX = *unknown* |
| | CX = *unknown* |
| | DX popped from stack |

### 40-52 386MAX v7.00 Function 27h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 27h *unknown* | AH = status (see Table 38-1) |
| AL = subfunction | DX popped from stack |
| 00h get *unknown* | BX = *unknown* |
| 01h *unknown* | |
| BX = *unknown* | |
| *ES* = *unknown* | |
| 02h *unknown* | |
| *unknown* | |
| 03h *unknown* | *unknown* |
| CX = *unknown* | |
| DX = *unknown* | |
| *ES* = *unknown* | |

### 40-53 386MAX v7.00 Function 28h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 28h get *unknown* | AH = status (see Table 38-1) |
| | CX = *unknown* |
| | DX = *unknown* |
| | STACK: top word popped |

### 40-54 386MAX v7.00 Function 29h calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 29h get *unknown* | AX = *unknown* |
| | DX popped from stack |

### 40-55 386MAX EMS services calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = 40h-5Dh EMS services (see INT 67h Function 40h etc) | as appropriate for EMS function |
| other registers as appropriate for EMS function | STACK popped (value placed in DX if no specific return value for DX) |
| STACK: WORD value to be passed in DX | |

### 40-56 386MAX VCPI services calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AH = DEh VCPI services (see INT 67h Function DE00h etc) | AH = status (see Table 38-1) |
| other registers as appropriate for VCPI function | STACK popped (value placed in DX if no specific return value for DX) |

### 40-57 Format of memory speed record:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | page table entry for 4K page |
| 04h | WORD | number of microticks (840ns units) required for REP LODSD of entire 4K page |

### 40-58 Bitfields for high memory information:

| | |
|---|---|
| bit 0 | *unknown* |
| bit 1 | physical address same as linear address |

## INT 21h Function 4402h

| | |
|---|---|
| bit 2 | EMS page frame |
| bit 6 | *unknown* |

**40-59 Format of memory info [array]:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | *unknown* |
| 04h | DWORD | *unknown* |
| 08h | 2 BYTEs | *unknown* |
| 0Ah | BYTE | *type* (03h = *conventional*) |
| 0Bh | BYTE | *unknown* |

### INT 21h Function 4403h, Subfunction 01h
### *TURN 386MAX OFF*

*386MAX v6.01+*
*(undocumented)*

**Registers at call:**
AX = 4403h
BX = handle for device "386MAX$$"
DS:DX -> BYTE 01h
CX ignored

**Return Registers:**
DS:DX -> BYTE status (00h = successful)
CF clear if successful
    AX destroyed
CF set on error
    AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Details:** This function will fail if any EMS or UMBs are in use.

**See also:** Function 4403h Subfunctions 02h, 03h, and 05h

### INT 21h Function 4403h, Subfunction 02h
### *TURN 386MAX ON*

*386MAX v6.01+*
*(undocumented)*

**Registers at call:**
AX = 4403h
BX = handle for device "386MAX$$"
DS:DX -> BYTE 02h
CX ignored

**Return Registers:**
DS:DX -> BYTE status (00h = successful)
CF clear if successful
    AX destroyed
CF set on error
    AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**See also:** Function 4403h Subfunctions 01h, 03h, and 05h

### INT 21h Function 4403h, Subfunction 03h
### *SET STATE*

*386MAX v6.01+*
*(undocumented)*

**Registers at call:**
AX = 4403h
BX = handle for device "386MAX$$"
CX = number of bytes to copy (up to size of state buffer)
DS:DX -> BYTE 03h followed by state buffer (see Table 40-1)

**Return Registers:**
CF clear if successful
    AX = number of bytes actually written
CF set on error
    AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**See also:** Function 4402h

### INT 21h Function 4403h, Subfunction 05h
### *LIMIT AUTOMATIC ACTIVATION TO STANDARD EMS CALLS*

*386MAX v7.00+*
*(undocumented)*

**Purpose:** Specify that 386MAX will only shift from AUTO to ON mode on standard EMS calls (INT 67h Functions 40h-5Dh).

**Registers at call:**
AX = 4403h
BX = handle for device "386MAX$$"

DS:DX -> BYTE 05h
CX ignored

**Return Registers:**
CF clear if successful
AX destroyed
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**See also:** Function 4403h Subfunctions 02h, 03h, and 06h

---

### INT 21h Function 4403h, Subfunction 06h
### *ALLOW AUTOMATIC ACTIVATION ON ANY INT 67h*

*386MAX v7.00+*
*(undocumented)*

**Purpose:** Specify that 386MAX should shift from AUTO to ON mode on any INT 67h call other than INT 67h Function 3Fh.

**Registers at call:**
AX = 4403h
BX = handle for device "386MAX$$"
DS:DX -> BYTE 06h
CX ignored

**Return Registers:**
CF clear if successful
AX destroyed
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**See also:** Function 4403h Subfunctions 01h, 03h, and 05h

---

### INT 67h Function 1Eh
### *MEMLIMIT - INSTALLATION CHECK*

*386MAX v7.00*
*(undocumented)*

**Registers at call:**
AH = 1Eh

**Return Registers:**
AH = 00h if installed
AL destroyed
ES:DI -> ASCII signature "MemLimit"

**Conflicts:** see Table 2-3
**See also:** Function 1Fh, INT 21h Function 4402h

---

### INT 67h Function 1Fh
### *MEMLIMIT - API*

*386MAX v7.00*
*(undocumented)*

**Registers at call:**
AH = 1Fh
DS:SI -> request packet (see Table 40-60)

**Return Registers:**
AH = status (see Table 38-1)

**Conflicts:** see Table 2-3
**See also:** Function 1Eh

**40-60 Format of request packet:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | function code (00h-0Fh) |
| 02h | WORD | return code (see Table 40-61) |
| 04h | 4 BYTEs | *unknown* |
| 08h | WORD | *unknown* |
| | *others, if any, unknown* | |

**40-61 Values for return code:**

| | |
|---|---|
| 00h | unknown request |
| 01h | invalid parameter for VCPI limit |
| 02h | VCPI limit set |
| 03h | invalid parameter for EMS limit |
| 04h | EMS limit set |
| 05h | DPMI disabled |
| 06h | XMS disabled |
| 07h | XMS limit set |
| 08h | unable to uninstall |
| 09h | unloaded |

## CHAPTER 41

# *Helix Netroom*

RAM-MAN/386 is the memory manager included with Helix's Netroom; MICEMM is a memory manager for some Micronics motherboards whose API is a subset of Netroom's.

---

**INT 16h Function 5758h, Subfunction 5754h** *Unknown Netroom program (undocumented)*
***UNDETERMINED FUNCTION***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5758h | *unknown* |
| BX = 5754h | |
| *others, if any, unknown* | |

---

**INT 16h Function 5758h, Subfunction 5754h** *Unknown Netroom program (undocumented)*
***UNDETERMINED FUNCTION***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5758h | *unknown* |
| BX = 5755h | |
| DS:SI -> *unknown* | |

---

**INT 16h Function 5758h, Subfunction 5756h** *Unknown Netroom program (undocumented)*
***INSTALLATION CHECK***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5758h | BX <> 5756h if installed |
| BX = 5756h | |

---

**INT 16h Function 5758h, Subfunction 5858h** *Netroom PRENET (undocumented)*
***GET OLD INTERRUPT VECTORS***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5758h | CF clear |
| BX = 5858h | DX:BX -> saved copy of interrupt vector table |

**Details:** The installation check consists of calling this function and comparing BX against 5858h on return; if it has changed, PRENET is installed. **See also:** Function 5758h Subfunction 5859h

---

**INT 16h Function 5758h, Subfunction 5859h** *Netroom POSTNET (undocumented)*
***GET OLD INTERRUPT VECTORS***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5758h | CF clear |
| BX = 5859h | DX:BX -> saved copy of interrupt vector table |

**Details:** The installation check consists of calling this function and comparing BX against 5859h on return; if it has changed, POSTNET is installed.
**See also:** Function 5758h Subfunction 5858h

## INT 2Fh Function 4310h

*RM386 v6.00*

### *XMS EXTENSIONS*

**Registers at call:**
AX = 4310h

**Return Registers:**
ES:BX -> driver entry point (see Tables 41-1 to 41-5)

**Details:** RM386 supports the standard XMS function calls (Tables 38-7 to 38-29) as well as its own private extensions shown below.

**See also:** XMS Function 4310h (Chapter 38)

#### 41-1 RM386 "Reallocate upper memory block" calling convention:

**Registers at call:**
AH = 80h
DX = segment address of UMB to resize
BX = new size of block in paragraphs

**Return Registers:**
AX = status
0001h success
0000h failure
BL = error code (80h,B0h,B2h) (see Table 38-30
DX = maximum available size

**Details:** This function is identical to XMS function 12h (see Table 38-25).

#### 41-2 RM386 "re-enable HMA allocation" calling convention:

**Registers at call:**
AH = 81h

**Return Registers:**
AX = 0001h (success)

#### 41-3 RM386 Cloaking API calling convention:

**Registers at call:**
AH = 82h
DX = XMS handle of block containing protected-mode code (see Table 41-7)
CL = code size (00h 16-bit, else 32-bit)
ESI, EDI = parameters to pass to protected-mode code

**Return Registers:**
AX = status
0001h success
0000h failed
BL = error code (A2h,B0h) (see Table 38-30)

**Details:** This function calls offset 0 in the XMS memory block as described in Table 41-7.

#### 41-4 RM386 "Create new UMB entry" calling convention:

**Registers at call:**
AH = 83h
BX = segment of high-memory block
DX = first page of start of block
CX = number of consecutive pages in block
DI = start of UMB in block

**Return Registers:**
AX = 0001h (success)
DI = segment of first high-DOS block

**Details:** The new UMB is not linked into the high-memory chain.

#### 41-5 RM386 "Get all XMS handles info" calling convention:

**Registers at call:**
AH = 84h
CX = size of buffer for handle info
ES:DI -> buffer for handle info (see Table 41-6)

**Return Registers:**
AX = 0001h (success)
DX = current number of allocated XMS handles

#### 41-6 Format of XMS handle information [array]:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | handle |
| 01h | BYTE | lock count |
| 02h | DWORD | handle size |
| 06h | DWORD | handle physical address (only valid if lock count nonzero) |

## INT 2Fh Function 4310h

**41-7 Cloaked code calling convention:**

EBX = physical address of block's start
CS = code selector for XMS block at EBX (16-bit or 32-bit)
DS = data selector for XMS block, starting at EBX
ES = selector for V86 memory access to full real-mode 1088K
GS = selector for full flat address space
ESI, EDI from V86 mode

**Return Registers:**
not relevant

---

### INT 31h Function 57h

Netroom3 DPMI.EXE v3.00

### *UNDETERMINED FUNCTION*

**Registers at call:**
AH = 57h
AL = subfunction (at least 02h,03h,04h,05h,07h,08h,09h,0Ah)
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Cloaking INT 2Ch Function 0000h (Chapter 32)

---

### INT 31h Function 5702h

Netroom3 DPMI.EXE v3.00

### *SWITCH TO PROTECTED MODE*

**Registers at call:**
AX = 5702h
DX = PSP segment of caller
STACK: WORD *unknown*
WORD flags (bit 0 set if 32-bit program)

**Return Registers:**
as for DPMI mode-switch entry point (see Table 30-1)

**Details:** This function is called by the real-mode DPMI mode-switch entry point.

**See also:** DPMI INT 2Fh Function 1687h (Chapter 30)

---

### INT 67h Function 56FFh

RM386 v6.00

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 56FFh
DS:SI -> *unknown*
*others, if any, unknown*

**Return Registers:**
*unknown*

**Conflicts:** see Table 2-3

---

### INT 67h Function 5857h

Netroom
(undocumented)

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = 5857h
BX = *function* (0057h,0059h,0159h seen)
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** BX=0059h appears to be analogous to Function 5800h and BX=0159h appears to be analogous to Function 5801h; BX=0057h appears to indicate whether AX=580xh or AX=5857h/BX=0x59h should be used.

**Conflicts:** see Table 2-3

**See also:** Function 5BF0h

---

### INT 67h Function 5BE0h

MICEMM v4D, RM386

### *GET LINEAR ADDRESS OF MEMORY*

**Registers at call:**
AX = 5BE0h
ES:BX -> memory for which to get linear address

**Return Registers:**
AH = 00h
CX:DX = linear address of physical memory corresponding to ES:BX

**Details:** This has been superceded by VCPI Function DE06h (Chapter 29), which should be used instead.
**Conflicts:** see Table 2-3
**See also:** Functions 5BF0h and 5BF1h, VCPI Function DE06h (Chapter 29)

---

### INT 67h Function 5BE1h
### *GET MEMORY MANAGER SIZE*

*RM386 v6.00+*

**Registers at call:**
AX = 5BE1h

**Return Registers:**
AH = 00h
CX = code and data size in bytes
DX:BX = physical address of RM386 code
DI:SI = total size of RM386 area including handle tables
BP = number of additional pages (high DOS, etc.)

**Conflicts:** see Table 2-3
**See also:** Functions 5BE0h and 5BE2h

---

### INT 67h Function 5BE2h
### *GET INTERRUPT VECTORS*

*RM386 v6.00+*

**Registers at call:**
AX = 5BE2h

**Return Registers:**
DS:SI -> V86-mode table (see Table 41-8)
ES:BX -> *unknown* (undocumented, middle of device driver interrupt routine!)

**Conflicts:** see Table 2-3
**See also:** Functions 5BE0h and 5BE1h

**41-8 Format of V86-mode table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | original INT 13h vector |
| 04h | DWORD | original INT 15h vector |
| 08h | DWORD | original INT 19h vector |
| 0Ch | DWORD | original INT 21h vector |
| 10h | DWORD | original INT 4Bh vector |
| 14h | DWORD | original INT 67h vector |

---

### INT 67h Function 5BF0h
### *INSTALLATION CHECK*

*MICEMM v4D, RM386*

**Registers at call:**
AX = 5BF0h

**Return Registers:**
AH = 00h if MICEMM or RM386 present
BX = code segment of driver

**Conflicts:** see Table 2-3
**See also:** Functions 5BE0h and 5BF1h, QEMM Function 3Fh (Chapter 39)

---

### INT 67h Function 5BF1h
### *GET ADDRESS MAP*

*MICEMM v4D, RM386*

**Registers at call:**
AX = 5BF1h
ES:BX -> 256-byte (MICEMM) or 512-byte (RM386) buffer for memory types

**Return Registers:**
AH = 00h
ES:BX buffer filled

**Details:** Each byte in the buffer specifies the type of a 4K page of memory.
**Conflicts:** see Table 2-3
**See also:** Functions 5BE0h and 5BF0h

**41-9 Values for memory type:**

| | |
|---|---|
| 00h | unused (MICEMM), RAM/available (RM386) |
| 02h | DOS extension (XMS UMB) |
| 04h | shadowed ROM |

| | |
|---|---|
| 08h | mappable EMS |
| 10h | page frame |
| 20h | ROM |
| 40h | reserved (video memory, etc) |
| 80h | RAM (MICEMM), Windows UMB (RM386) |

---

## INT 67h Function 5BF2h

*RM386*

### *GET RM386 INTERNAL DATA*

| Registers at call: | Return Registers: |
|---|---|
| AX = 5BF2h | buffer filled |
| CX = size of buffer | |
| DS:SI -> buffer for internal data | |

(documentation says ES:BX -> buffer, SI = offset within RM386)

**Details:** The data returned by this function is release-specific.

**Conflicts:** see Table 2-3

**See also:** Function 5BF0h

---

## INT 67h Function 5BF3h

*RM386*

### *RETURN TO REAL MODE*

| Registers at call: | Return Registers: |
|---|---|
| AX = 5BF3h | none |

**Details:** This function is considered obsolete; use Function 5DE0h instead.

**Conflicts:** see Table 2-3

**See also:** Functions 5BF0h and 5DE0h

---

## INT 67h Function 5BF4h

*RM386 v6.00*

### *GET RM386 GLOBAL FLAGS*

| Registers at call: | Return Registers: |
|---|---|
| AX = 5BF4h | AH = 00h |
| | BX = global flags 1 (see Table 41-10) |
| | CX = global flags 2 (see Table 41-11) |
| | DX = global flags 3 (see Table 41-12) |
| | SI = global flags 4 (see Table 41-13) |

**Conflicts:** see Table 2-3

**See also:** Function 5BF0h

**41-10 Bitfields for global flags 1:**

| | |
|---|---|
| bits 0-3 | reserved |
| bit 4 | V86 mode |
| bit 5 | reserved |
| bit 6 | 80386 or higher CPU |
| bits 7,8 | reserved |
| bit 9 | A20 enabled at startup |
| bit 10 | “HIGH_IO” |
| bit 11 | ROM |
| bit 12 | large frame |
| bits 13,14 | reserved |
| bit 15 | PS/2-style A20 control |

**41-11 Bitfields for global flags 2:**

| | |
|---|---|
| bit 0 | HMA in use |
| bit 1 | XMS present |
| bit 2 | using XMS driver memory |

| | |
|---|---|
| bit 3 | HIGH (NEAT only) |
| bits 4-7 | reserved |
| bit 8 | NOBKTRAP |
| bit 9 | NORESET |
| bit 10 | ALTMAP |
| bit 11 | NOFRAME |
| bits 12-15 | reserved |

**41-12 Bitfields for global flags 3:**

| | |
|---|---|
| bit 0 | NOTEST |
| bit 1 | NOEBDA |
| bit 2 | Windows3 support |
| bit 3 | system board mouse |
| bit 4 | DISKBUF |
| bit 5 | EBDALOW |
| bit 6 | A20 global enable flag |
| bit 7 | A20 flag |
| bit 8 | EBDA moved to stub |
| bit 9 | VXD file was found |
| bit 10 | reserved |
| bit 11 | NOBOOTMAP |
| bit 12 | AUTO |
| bit 13 | PS/2 machine |
| bit 14 | Compaq ROM merge active |
| bit 15 | NOHMA set |

**41-13 Bitfields for global flags 4:**

| | |
|---|---|
| bit 0 | "NOV8259" don't virtualize interrupt controller |
| bit 1 | NOSCSI |
| bit 2 | NOSCAN |
| bit 3 | NOTR |
| bit 4 | ALTBOOT |
| bit 5 | NOCOMPQ |
| bit 6 | KB2TRAP |
| bit 7 | DESHADOW |
| bit 8 | Video 7 VGA detected |
| bit 9 | reserved |
| bit 10 | NOVGA |
| bit 11 | NOPS2 |
| bit 12 | DEBUG |
| bit 13 | NOVKB |
| bits 14,15 | reserved |

---

## INT 67h Function 5BF5h
## *GET RM386 EMS HANDLE COUNT*

*RM386 v6.00*

**Registers at call:**
AX = 5BF5h

**Return Registers:**
AH = status
 00h successful
  BX = current number of allocated EMS handles
 84h function not available

**Conflicts:** see Table 2-3
**See also:** Function 5BF0h

---

## INT 67h Function 5DE0h *RM386*

### *DISABLE RM386*

**Registers at call:**
AX = 5DE0h

**Return Registers:**
none

**Details:** RM386 traps this function on the initial transition to protected mode caused by the INT instruction, which means it can not be overridden simply by hooking the interrupt.

**Conflicts:** see Table 2-3

**See also:** Function 5DE1h

---

## INT 67h Function 5DE1h *RM386*

### *ENABLE RM386*

**Registers at call:**
AX = 5DE1h

**Return Registers:**
none

**Details:** RM386 traps this function on the initial transition to protected mode caused by the INT instruction, which means it can not be overridden simply by hooking the interrupt.

**Conflicts:** see Table 2-3

**See also:** Function 5DE0h

---

## INT 67h Function 5DE2h *RM386*

### *GET PAGE TABLE*

**Registers at call:**
AX = 5DE2h
ES:DI -> 1088-byte buffer for page table

**Return Registers:**
ES:DI buffer filled

**Details:** RM386 traps this function on the initial transition to protected mode caused by the INT instruction, which means it can not be overridden simply by hooking the interrupt.

**Conflicts:** see Table 2-3

**See also:** Function 5DE3h

---

## INT 67h Function 5DE3h *RM386*

### *SET PAGE TABLE ACCESS BITS*

**Registers at call:**
AX = 5DE3h
ES:DI -> 1088-byte buffer containing page table

**Details:** Only the access bits of the page table are used, the remainder is ignored.

RM386 traps this function on the initial transition to protected mode caused by the INT instruction, which means it can not be overridden simply by hooking the interrupt.

**Conflicts:** see Table 2-3

**See also:** Function 5DE2h

---

## INT 67h Function 5DE4h *RM386*

### *SET WRITE-PROTECTION FOR PAGE IN FIRST MEGABYTE*

**Registers at call:**
AX = 5DE4h
BL = page number
BH = access (00h read-only, 01h read-write)

**Details:** RM386 traps this function on the initial transition to protected mode caused by the INT instruction, which means it can not be overridden simply by hooking the interrupt.

**Conflicts:** see Table 2-3

## INT 67h Function 5DE5h

*RM386*

### *MAP PHYSICAL PAGE TO PHYSICAL SEGMENT*

**Registers at call:**
AX = 5DE5h
EBX = physical page number
DX = page number in first megabyte to be remapped (linear address SHR 12)

**Return Registers:**
AH = status (00h,8Bh) (see Table 41-14)

**Details:** RM386 traps this function on the initial transition to protected mode caused by the INT instruction, which means it can not be overridden simply by hooking the interrupt.

**Conflicts:** see Table 2-3

**See also:** Function 5DE6h

**41-14 Values for RM386 status:**

| | |
|---|---|
| 00h | successful |
| 83h | invalid handle |
| 8Ah | invalid logical page (out of handle's range) |
| 8Bh | invalid destination page (not in first megabyte) |
| A5h | invalid page bits |
| A6h | invalid page count (overflows first megabyte) |
| A7h | invalid port ID |
| A8h | reserved port--cannot trap/untrap (DMA/INT/KBD controllers) |

## INT 67h Function 5DE6h

*RM386*

### *MAP LOGICAL 4K PAGE TO PHYSICAL SEGMENT*

**Registers at call:**
AX = 5DE6h
BX = logical page number in 4K pages from beginning of memory for EMS handle
CX = segment in first megabyte to be remapped
DX = previously-allocated EMS handle

**Return Registers:**
AH = status (00h,83h,8Ah,8Bh) (see Table 41-14)

**Details:** RM386 traps this function on the initial transition to protected mode caused by the INT instruction, which means it can not be overridden simply by hooking the interrupt.

**Conflicts:** see Table 2-3

**See also:** Function 5DE5h

## INT 67h Function 5DE7h

*RM386*

### *SET PAGE TABLE BITS FOR RANGE OF PAGES*

**Registers at call:**
AX = 5DE7h
BL = page table bits to be set (bits 2-0 = U/S, R/W, P)
CX = number of pages to set
DX = first page number to set (in first megabyte)

**Return Registers:**
AH = status (00h,8Bh,A5h,A6h) (see Table 41-14)

**Details:** RM386 traps this function on the initial transition to protected mode caused by the INT instruction, which means it can not be overridden simply by hooking the interrupt.

**Conflicts:** see Table 2-3

## INT 67h Function 5DE8h

*RM386*

### *GET PARTIAL PAGE TABLE*

**Registers at call:**
AX = 5DE8h
BX = starting page number in first

megabyte+HMA (0000h-010Fh)
CX = number of page table entries to get
ES:DI -> buffer for DWORD page table entries

**Return Registers:**
AH = status (00h,8Bh) (see Table 14-14)

**Details:** RM386 traps this function on the initial transition to protected mode caused by the INT instruction, which means it can not be overridden simply by hooking the interrupt.
**Conflicts:** see Table 2-3
**See also:** Function 5DE9h

---

### INT 67h Function 5DE9h *RM386*
### *SET PARTIAL PAGE TABLE*

**Registers at call:**
AX = 5DE9h
BX = starting page number in first megabyte+HMA (0000h-010Fh)
CX = number of page table entries to get
DS:SI -> buffer of DWORD page table entries

**Return Registers:**
AH = status (00h,8Bh) (see Table 41-14)

**Details:** RM386 traps this function on the initial transition to protected mode caused by the INT instruction, which means it can not be overridden simply by hooking the interrupt.
**Conflicts:** see Table 2-3
**See also:** Function 5DE8h

---

### INT 67h Function 5DEAh *RM386*
### *V86-MODE I/O PORT TRAPPING CONTROL*

**Registers at call:**
AX = 5DEAh
BX = function
  00h globally disable V86-mode trapping
  01h globally enable V86-mode trapping
    CL = interrupt to use for trapping
  02h get I/O trapping state

**Return Registers:**
AH = status
  00h successful
    BX = current trapping state (function 02h)
      0000h disabled, 0001h enabled
    CX = interrupt used as trap interrupt (functions 00h and 02h)

**Details:** RM386 traps this function on the initial transition to protected mode caused by the INT instruction, which means it can not be overridden simply by hooking the interrupt.

When I/O trapping is enabled and I/O port access occurs, RM386 simulates an INT instruction for the specified interrupt; the interrupt handler is responsible for decoding the trapped instruction and performing the appropriate action. INT 2Ch Function 002Dh provides a similar but more-easily used interface.
**Conflicts:** see Table 2-3
**See also:** Functions 5DEBh and EFh, Cloaking INT 2Ch Function 002Dh (Chapter 32)

---

### INT 67h Function 5DEBh *RM386*
### *V86-MODE I/O TRAPPING PORT CONTROL*

**Registers at call:**
AX = 5DEBh
BX = function
  00h disable V86-mode trapping for specified port
  01h enable V86-mode trapping for specified port
  02h get V86-mode trapping state for specified port
DX = port for which to enable/disable/query trapping

**Return Registers:**
AH = status (00h,A7h,A8h) (see also Table 41-14)
  00h successful
    BX = current trapping state (00h off, 01h on) (function 02)

**Details:** RM386 traps this function on the initial transition to protected mode caused by the INT instruction, which means it can not be overridden simply by hooking the interrupt.
**Conflicts:** see Table 2-3
**See also:** Function 5DEAh

---

## INT 67h Functions 5DFDh to 5DFFh
### *UNDETERMINED FUNCTIONS*

*RM386 v6.00 (undocumented)*

**Registers at call:**
AX = 5DFDh to 5DFFh
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** RM386 traps these functions on the initial transition to protected mode caused by the INT instruction, which means they can not be overridden simply by hooking the interrupt.
**Conflicts:** see Table 2-3

---

## INT 67h Function DE0Fh
### *UNDETERMINED FUNCTION*

*Netroom3*

**Registers at call:**
AX = DE0Fh
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** This function is called by Netroom's DPMI.EXE.
**Conflicts:** see Table 2-3

---

## INT 67h Function EFh
### *EXECUTE XMS FUNCTION*

*RM386 v6.00+*

**Registers at call:**
AH = EFh
AL = function (00h-12h,80h-8Fh)
other registers as appropriate for XMS function

**Return Registers:**
varies by function (see Tables 38-7 to 38-29)

**Details:** These functions appear to be equivalent to the XMS functions with the same numbers.
**Conflicts:** see Table 2-3
**See also:** Function 5Dh, XMS INT 2Fh Function 4310h (Chapter 38)

---

## INT 7Eh Function 5857h, Subfunction 57h
### *INSTALLATION CHECK*

*XLOAD*

**Purpose:** XLOAD is the LOADHIGH-equivalent from Helix Software's Netroom.

**Registers at call:**
AX = 5857h
BL = 57h

**Return Registers:**
AX = 0000h if installed

**Conflicts:** GO32 (Chapter 37), YTERM (Chapter 66)
**See also:** Function 5857h Subfunctions 5Ah and 5Bh

---

## INT 7Eh Function 5857h, Subfunction 5Ah
### *GET UNKNOWN VALUE*

*XLOAD*

**Registers at call:**
AX = 5857h
BL = 5Ah

**Return Registers:**
AX = 0000h
CX = *unknown*
DS:SI -> *unknown*

**Conflicts:** GO32 (Chapter 37), YTERM (Chapter 66)
**See also:** Function 5857h Subfunctions 57h and 5Bh

---

### INT 7Eh Function 5857h, Subfunction 5Bh

*XLOAD*

### *GET XLOAD MEMORY SIZE*

| Registers at call: | Return Registers: |
|---|---|
| AX = 5857h | AX = 0000h |
| BL = 5Bh | CX = segment of XLOAD TSR |
| | DX = size of memory block in which XLOAD TSR is located |

**Conflicts:** GO32 (Chapter 37), YTERM (Chapter 66)
**See also:** Function 5857h Subfunctions 57h and 5Ah

---

### INT B5h

*Netroom NETSWAP4 (undocumented)*

### *UNDETERMINED FUNCTION*

| Registers at call: | Return Registers: |
|---|---|
| *unknown* | *unknown* |

**See also:** INT 31h Function 57h

## CHAPTER 42

# *Other Memory Managers*

This chapter covers those memory managers with fairly small APIs that do not justify separate chapters. The first section in this chapter covers several calls which are supported by multiple memory managers, for MS Windows compatibility. The chapter continues with sections on Microsoft's HIMEM and EMM386, Compaq's CEMM, Key Software Product's The Last Byte, Nanosoft's MD386, and three miscellaneous calls.

## Common Calls

A number of memory managers have recently adopted some of the calls provided by Microsoft's EMM386, to achieve compatibility with the MS Windows interface for cooperation with memory managers.

---

### INT 21h Function 4402h, Subfunction 00h

***GET API ENTRY POINT***

*Memory Managers (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4402h | CF clear if successful<br>buffer filled (see Table 42-1) |
| BX = file handle for device "EMMXXXX0" | CF set on error |
| CX = 0006h (size of buffer in bytes) | AX = error code (01h,05h,06h,0Dh)<br>(see Table 24-2) |
| DS:DX -> buffer for API entry point record<br>first byte must be 00h on entry | |

**Details:** This function is supported by Microsoft EMM386.EXE v4.45+ and CEMM v5.10+.

If no other program has hooked INT 67h, an alternate installation check for CEMM is testing for the string "COMPAQ EXPANDED MEMORY MANAGER 386" at offset 14h in the INT 67h handler's segment; if present, the word at offset 12h contains the offset of the API entry point.

**See also:** Function 4402h Subfunctions 01h and 02h, EMM386 Function 4402h (below), QEMM INT 67h Function 3Fh (Chapter 39)

**42-1 Format of API entry point record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | *unknown* (0022h for CEMM 5.11, 0025h for MS EMM386 v4.45) |
| 02h | DWORD | manager's private API entry point (see Tables 42-2 and 42-9) |

**42-2 CEMM v5.10+ entry point calling convention:**

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 00h get memory manager's state | AH = state<br>bit 0: turned OFF<br>bit 1: AUTO mode enabled |

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 01h set memory manager's state<br>AL = new state (00h ON, 01h OFF, 02h AUTO) | CF clear if successful<br>CF set on error |
| **Registers at call:** | **Return Registers:** |
| AH = 02h Weitek coprocessor support<br>AL = subfunction | CF clear if successful<br>CF set on error<br>AH = error code (01h invalid subfunc, 02h no Weitek) |
| 00h get Weitek support state | AL = status<br>bit 0: Weitek coprocessor is present<br>bit 1: Weitek support is enabled |
| 01h turn on Weitek support<br>02h turn off Weitek support | |
| **Registers at call:** | **Return Registers:** |
| AH = 05h get statistics<br>*others, if any, unknown* | *unknown* |
| **Registers at call:** | **Return Registers:** |
| AH > 06h | CF set<br>AH = 01h (invalid function) |

**Details:** AH=03h,04h,06h are NOPs which return CF clear, presumably for backwards compatibility with earlier versions of CEMM.

In v5.11, AH=05h merely prints an error message (using INT 21h Function 09h) stating that a different version of CEMM is installed and it is therefore not possible to display the statistics.

---

## INT 21h Function 4402h, Subfunction 01h
## *GET EMM IMPORT STRUCTURE ADDRESS*

*Memory Managers (undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4402h<br>BX = file handle for device "EMMXXXX0"<br>CX = 0006h (size of buffer in bytes)<br>DS:DX -> buffer for EMM import structure record<br>first byte must be 01h on entry | CF clear if successful<br>buffer filled (see Table 42-3)<br>CF set on error<br>AX = error code (01h,05h,06h,0Dh) (see Table 24-2) |

**Details:** This function is supported by Microsoft EMM386.EXE v4.45+, QEMM-386 v6+, and CEMM v5.10+.

For QEMM-386, this call always returns an error if Windows3 support has been disabled with the NW3 switch.

**See also:** Function 4402h Subfunction 00h, EMM386 Function 4402h (below), QEMM INT 2Fh Function D201h Subfunction 5145h (Chapter 39), QEMM INT 67h Function 3Fh (Chapter 39)

**42-3 Format of EMM import structure record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | physical address of EMM import structure |
| 04h | BYTE | EMM import structure major version |
| 05h | BYTE | EMM import structure minor version |

### INT 21h Function 4402h, Subfunction 02h
### *GET MEMORY MANAGER VERSION*

*Memory Managers*
*(undocumented)*

**Registers at call:**
AX = 4402h
BX = file handle for device "EMMXXXX0"
CX = 0002h (size of buffer in bytes)
DS:DX -> buffer for memory manager version (see Table 42-4)
first byte must be 02h on entry

**Return Registers:**
CF clear if successful
buffer filled
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Details:** This function is supported by Microsoft EMM386.EXE v4.45+ and CEMM v5.10+.

**See also:** Function 4402h Subfunction 00h, EMM386 Function 4402h, QEMM INT 67h Function 3Fh (Chapter 39)

**42-4 Format of memory manager version:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | major version |
| 01h | BYTE | minor version (binary) |

## Microsoft HIMEM and EMM386

HIMEM.SYS and EMM386.EXE are the two parts of Microsoft's own memory management capability, furnished with Windows and MS-DOS (DR-DOS and Novell DOS also provide drivers of the same or similar names, but this section deals only with the Microsoft products).

HIMEM.SYS is an extended-memory manager, that is, a program that coordinates the use of extended memory, including the high memory area (HMA), so that no two processes attempt to use the same memory at the same time. EMM386.EXE uses extended memory (XMS) to simulate expanded memory (EMS) for programs that can use EMS. EMM386 also makes it possible to load programs and device drivers into upper memory blocks (UMBs).

### INT 21h Function 4402h
### *GET MEMORY MANAGER INFORMATION*

*Microsoft EMM386.EXE v4.45*
*(undocumented)*

**Registers at call:**
AX = 4402h
BX = file handle for device "EMMXXXX0"
CX = size of buffer in bytes (varies, see Table 42-5)
DS:DX -> buffer for returned data (see Table 42-5)
first byte must be set on entry to indicate desired data

**Return Registers:**
CF clear if successful
buffer filled
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**Details:** An error is returned if the number of bytes to be read does not match the number of bytes returned for the specified data item.

**See also:** INT 67h Function FFA5h , Function 4402h Subfunctions 00h to 02h (above)

**42-5 Format of data buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | (call) function<br>03h get *unknown value*<br>04h get *unknown value* |
| ---function 03h--- | | |
| 00h | WORD | *unknown* |
| 02h | WORD | *unknown* |
| ---function 04h--- | | |
| 00h | WORD | segment of UMB containing EMM386 code/data |
| 02h | WORD | number of paragraphs of EMM386 code/data in UMB |
| 04h | WORD | *unknown* |

### INT 2Fh Function 4308h
### *GET A20 HANDLER NUMBER*

*HIMEM.SYS v2.77+*
*(undocumented)*

**Registers at call:**
AX = 4308h

**Return Registers:**
AL = 43h if supported
BL = A20 handler number (value of /MACHINE:nn switch)
BH = AT A20 switch time (00h medium, 01h fast, 02h slow)

**See also:** Functions 4309h and 4330h

### INT 2Fh Function 4309h
### *GET XMS HANDLE TABLE*

*HIMEM.SYS v3.09+*
*(undocumented)*

**Registers at call:**
AX = 4309h

**Return Registers:**
AL = 43h if function supported
ES:BX -> XMS handle table (see Table 42-6)

**Details:** HIMEM.SYS v3.09 is part of MS-DOS 6.0.
**See also:** Function 4308h

**42-6 Format of XMS handle table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | *unknown* (01h in HIMEM.SYS v3.09) |
| 01h | BYTE | *size of one handle* (0Ah in HIMEM.SYS v3.09) |
| 02h | WORD | value of /numhandles= parameter (default = 20h) |
| 04h | DWORD | pointer to XMS handle array (see Table 42-7) |

**42-7 Format of XMS handle [array]:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | flag 01h=free, 02h=used, 04h=unused |
| 01h | BYTE | lock 00h=unlocked, 01h=locked |
| 02h | DWORD | address of XMS block in KB (shift left by 10 for abs. address) |
| 06h | DWORD | size of XMS block in KB |

### INT 2Fh Function 4320h
### *Mach 20 SUPPORT*

*HIMEM.SYS*
*(undocumented)*

**Registers at call:**
AX = 4320h
*others, if any, unknown*

**Return Registers:**
*unknown*

### INT 2Fh Function 4330h
### *GET EXTERNAL A20 HANDLER ADDRESS*

*HIMEM.SYS v2.77+*
*Callout (undocumented)*

**Registers at call:**
AX = 4330h

**Return Registers:**
AL = 80h if external A20 handler provided
ES:BX -> external A20 handler (see Table 42-8)
CL = *unknown*

**Details:** HIMEM.SYS calls this function to allow an external program to provide an A20 handler (i.e. to support a machine not supported by HIMEM itself).
**See also:** Function 4308h, XMS Function 4310h (Chapter 38)

**42-8 External A20 handler calling convention:**

**Registers at call:**
AX = function
0000h disable A20
0001h enable A20

**Return Registers:**
AX = status
0000h failure
0001h successful

### INT 67h Function FFA5h

*Microsoft EMM386.EXE v4.20+*

### *INSTALLATION CHECK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = FFA5h | AX = 845Ah if loaded<br>BX:CX -> API entry point (see Table 42-9) |

**Details:** This call is available even if EMM386 is not providing EMS.

If no other program has hooked INT 67h, an alternate installation check is to test for the string "MICROSOFT EXPANDED MEMORY MANAGER 386" at offset 14h in the INT 67h handler's segment; the word immediately preceding this string contains the offset of the API entry point.

**Conflicts:** see Table 2-3

**See also:** Function FFA5h Subfunction 4345h, INT 21h Function 4402h, QEMM Function 3Fh (Chapter 39)

---

**42-9 API entry point calling convention:**

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 00h get memory manager's status | AH = status<br>bit 0: not active (OFF)<br>bit 1: in "Auto" mode |

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 01h set memory manager's state<br>AL = new state (00h ON, 01h OFF, 02h AUTO) | none |

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 02h Weitek coprocessor support<br>AL = subfunction | |
| 00h get Weitek support state | AL = status<br>bit 0: Weitek coprocessor is present<br>bit 1: Weitek support is enabled |
| 01h turn on Weitek support<br>02h turn off Weitek support | |

--- **v4.20-4.41 only** ---

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 03h *Windows support*<br>AL = subfunction (00h, 01h) | none |

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 04h print copyright notice to standard output (using INT 21h Function 09h) | none |

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 05h print available report (the one shown when running EMM386 from the DOS prompt) | none |

## Compaq CEMM

CEMM is the expanded memory manager furnished by Compaq for its systems, which differ enough in architecture to make other memory manager programs less than fully reliable in all cases.

---

### INT 67h Function FFA5h, Subfunction 4345h
### *PRIVATE API*

*Compaq CEMM v5.10+*
*(undocumented)*

**Registers at call:**
AX = FFA5h
BX = 4345h ("CE")
DX = subfunction
0000h *unshadow video ROM*
0001h *shadow video ROM*
0002h map pages
CX = number of pages (00h=one)
ESI = linear address of first page to map into address space
EDI = linear starting address at which pages are to be visible
0003h get *unknown*
0004h BUG: crashes system due to fencepost error

**Return Registers:**
AH = 84h
AL = status (84h = error, FFh = success)
DX = *unknown* (0-2)

**Details:** If BX <> 4345h or DX > 0004h on entry, CEMM behaves identically to Microsoft's EMM386 (see Function FFA5h above).
**Conflicts:** see Table 2-3
**See also:** EMM386 Function FFA5h

## The Last Byte

"The Last Byte" is a shareware memory management package by Key Software Products. Among other programs, it consists of LASTBYTE.SYS and HIGHUMM.SYS.

---

### INT 21h Function 4402h
### *IOCTL - GET* UNKNOWN *TABLE*

*LASTBYTE.SYS v1.19*
*(undocumented)*

**Registers at call:**
AX = 4402h
BX = handle for device "LA$TBYTE"
CX = 0004h
DS:DX -> DWORD to hold address of 39-byte table of *unknown*

**Return Registers:**
CF set on error
AX = error code (see Table 24-2)
CF clear if successful
AX = number of bytes read

**See also:** HIGHUMM.SYS Function 4402h below

---

### INT 21h Function 4402h
### *IOCTL - GET API ADDRESS*

*HIGHUMM.SYS v1.17+*

**Registers at call:**
AX = 4402h
BX = handle for device "KSP$UMM"
CX = 0004h
DS:DX -> DWORD to hold entry point (see Table 42-10)

**Return Registers:**
CF set on error
AX = error code (see Table 24-2)
CF clear if successful
AX = number of bytes read

**Details:** Only API functions 00h and 01h are always available; the remaining functions are only enabled if the proper commandline switch is given.
**See also:** LASTBYTE.SYS Function 4402h above

**42-10 HIGHUMM.SYS entry point calling convention:**

**Registers at call:**
AH = 00h allocate UMB (see also Table 38-23)
DX = size in paragraphs

**Return Registers:**
AX = status code
0001h successful
BX = segment number

0000h failed
BL = error code (see Table 38-30)
DX = size of requested block/size of largest block

**Registers at call:**
AH = 01h deallocate UMB (see also Table 38-24)
DX = segment number of UMB

**Return Registers:**
AX = status code
0001h successful
0000h failed
BL = error code (see Table 38-30)

**Registers at call:**
AH = 02h request a bank-switched memory block
DX = size in paragraphs

**Return Registers:**
AX = status code
0001h successful
BX = segment number
0000h failed
BL = error code (see Table 38-30)
DX = size of requested block/size of largest block

**Registers at call:**
AH = 03h release a bank-switched memory block
DX = segment number

**Return Registers:**
(same as API function 01h)

**Registers at call:**
AH = 04h transfer data to/from high memory
DS:SI -> source
ES:DI -> destination
CX = length in bytes

**Return Registers:**
AX = status code
0001h successful
0000h failed
BL = error code (see Table 38-30)

**Details:** This function enables bank-switched memory, does the copy, then disables bank-switched memory.

**Registers at call:**
AH = 05h get a word from bank-switched memory
ES:DI -> word to read

**Return Registers:**
AX = status code
0001h successful
DX = word read
0000h failed
BL = error code (see Table 38-30)

**Registers at call:**
AH = 06h put a word to bank-switched memory
ES:DI -> word to write
DX = new value for word

**Return Registers:**
AX = status code
0001h successful
0000h failed
BL = error code (see Table 38-30)

**Registers at call:**
AH = 07h put a byte to bank-switched memory
ES:DI -> byte to write
DL = new value for byte

**Return Registers:**
(same as API function 06h)

**Registers at call:**
AH = 08h enable bank-switched memory
AH = 09h disable bank-switched memory
DS:SI -> 6-byte status save area

**Return Registers:**
AX = status code
0001h successful
0000h failed
BL = error code (see Table 38-30)

**Registers at call:**
AH = 0Ah assign name to UMB or high bank-switched block
DX = segment number
DS:SI -> 8-byte blank-padded name
**Registers at call:**
AH = 0Bh locate UMB block by name
AH = 0Ch locate bank-switched block by name
DS:SI -> 8-byte blank-padded name

**Return Registers:**
AX = status code
0001h successful
0000h failed
BL = error code (see Table 38-30)
**Return Registers:**
AX = status code
0001h successful
BX = segment number
DX = size of block
0000h failed
BL = error code (see Table 38-30)

## Nanosoft MD386

MD386 is a subset EMS memory manager by Nanosoft specifically designed for use with the MultiDOS Plus multitasker. Though not specifically listed with each entry, there are potential INT 67h conflicts with some hard disk controllers; see Table 2-3.

---

### INT 67h Function 5D03h
### *INTERNAL INITIALIZATION*

*Nanosoft MD386*
*(partially doc.)*

**Registers at call:**
AX = 5D03h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Functions 5D04h and 5E00h

---

### INT 67h Function 5D04h
### *GET ALTERNATE MAP STRUCTURE*

*Nanosoft MD386*

**Registers at call:**
AX = 5D04h
BX = alternate register set number
ES:DI -> 1024-byte buffer for map structure

**Return Registers:**
AH = status (see Table 38-1)
buffer filled if AH=00h

**Details:** This function is used for debugging purposes.
**See also:** Function 5D05h

---

### INT 67h Function 5D05h
### *GET INTERNAL HANDLE TABLE*

*Nanosoft MD386*

**Registers at call:**
AX = 5D05h
BX = handle number
ES:DI -> 1024-byte buffer for Handle table

**Return Registers:**
AH = status (see Table 38-1)
buffer filled if AH=00h

**Details:** This function is used for debugging purposes.
**See also:** Function 5D04h

---

### INT 67h Function 5E00h
### *SET HARDWARE BREAKPOINT*

*Nanosoft MD386*

**Registers at call:**
AX = 5E00h
DH = breakpoint number (0-3)
DL = breakpoint attributes (used to set DR7)
CX:BX = linear address of breakpoint
**See also:** Functions 5D03h and 5E01h

---

### INT 67h Function 5E01h
*Nanosoft MD386*

***GET HARDWARE DEBUG REGISTER***

**Registers at call:**
AX = 5E01h
BL = register number (0-3,7)

**Return Registers:**
CX:BX = value of specified DRx register

**See also:** Function 5E00h

---

### INT 67h Function 5E02h
*Nanosoft MD386*

***SET DEBUG EXCEPTION HANDLER***

**Registers at call:**
AX = 5E02h
CX:BX -> exception handler

**Details:** The specified exception handler is called with a simulated interrupt whenever a debug exception occurs which was caused by a hardware breakpoint set with the debug registers.

**See also:** Functions 5E04h and 5E05h

---

### INT 67h Function 5E03h
*Nanosoft MD386*

***ENABLE/DISABLE MEMORY WRITE PROTECTION***

**Registers at call:**
AX = 5E03h
BL = register map set number
BH = new state (00h read-only, else read-write)
CX = linear page number (linear address SHR 12)

**Details:** Setting write protection in map set 0 will cause the setting to become the default for newly-allocated map sets.

---

### INT 67h Function 5E04h
*Nanosoft MD386*

***GET DEBUG EXCEPTION***

**Registers at call:**
AX = 5E04h

**Return Registers:**
BL = debug exception (low byte of DR6 register)

**See also:** Functions 5E02h and 5E05h

---

### INT 67h Function 5E05h
*Nanosoft MD386*

***IGNORE NEXT DEBUG EXCEPTION***

**Registers at call:**
AX = 5E05h

**Details:** This function may be required when using Function 5E02h for handling instruction breakpoints.

**See also:** Functions 5E02h and 5E04h

## Miscellaneous

This section provides all information we have been able to locate for four lesser-known memory managers: Tandon's memory mapper, Biologic's HRAM, Quadtel's QMAPS, and Hewlett-Packard's HPMM.

---

### INT 15h Function F200h
*Tandon memory mapper*

***MAPPER HARDWARE INITIALIZATION CHECK***

**Registers at call:**
AX = F200h
CX = 454Dh

**Return Registers:**
CF clear if hardware already initialised
    BX = upper RAM areas in use
        bit 0: C000-C3FF
        bit 1: C400-C7FF
        ...
        bit 11: EC00-EFFF
CF set if hardware not initialised yet

### INT 2Fh Function A189h
### *API*

*Biologic HRAMDEV.SYS*
*(undocumented)*

**Purpose:** HRAMDEV.SYS is a part of the shareware package HRAM by Biologic which provides improved high memory access under MS-DOS 5.0.

**Registers at call:**
AX = A189h
BX = subfunction
0000h set *unknown*
0001h remove *unknown*
0002h get status *unknown*
0003h enable *unknown*
0004h disable *unknown*
0005h set *unknown* flag
0006h clear *unknown* flag
0007h set *unknown* flag
0008h clear *unknown* flag
0009h set *unknown*
ES:DI -> function-specific arguments
if func 0000h: 20-byte buffer containing *unknown*
if func 0001h: 20-byte buffer for returned *unknown*
if func 0002h: 16-byte buffer for returned *unknown*
if func 0009h: WORD containing *unknown*

**Return Registers:**
BX = A189h if installed
AH = status
00h successful
FFh failed or invalid function number

**Details:** Subfunctions 00h and 01h use a stack of four entries; Subfunction 01h always removes the values stored with the most recent Subfunction 00h call which has not yet been matched with a Subfunction 01h call.

**Conflicts:** Ergo DOS extenders (Chapter 34)

### INT 67h Function DDh
### *API*

*Quadtel QMAPS*

**Registers at call:**
AH = DDh
AL = function
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Details were not available at the time of writing. Hewlett-Packard's HPMM.SYS is a licensed version of QMAPS, and thus supports this API.

**Conflicts:** see Table 2-3

**See also:** EMM386 Function FFA5h (above), QEMM Function 3Fh (Chapter 39)

## CHAPTER 43

# *Microsoft Windows*

Microsoft Windows makes use of a number of interrupts for communication between its various modules, and for communicating with non-Windows programs that may be running as a task. This chapter describes those interrupt usages.

## Windows Interrupt Calls

INT 2Fh Functions 16XXh comprise an API for non-Windows programs (DOS device drivers, TSRs, and applications) to cooperate with multitasking Windows/386 2.x and Windows 3.x and higher enhanced mode. Certain of these calls are also supported in the Microsoft 80286 DOS extender in Windows standard mode.

---

### INT 2Eh UPh

*Windows NT*

***NATIVE API***

**Registers at call:**
EAX = function number
EDX = address of parameter block

**Return Registers:**
*unknown*

---

### INT 2Fh Function 1600h

*MS Windows*

***WINDOWS ENHANCED MODE INSTALLATION CHECK***

**Registers at call:**
AX = 1600h

**Return Registers:**
AL = status
  00h neither Windows 3.x enhanced mode nor Windows/386 2.x running
  01h Windows/386 2.x running
  80h XMS version 1 driver installed (neither Windows 3.x enhanced mode nor Windows/386 2.x running) (obsolete--see note)
  FFh Windows/386 2.x running
AL = anything else
  AL = Windows major version number >= 3
  AH = Windows minor version number

**Details:** This function served as the installation check and Function 1610h served to get the driver entry point for XMS version 1, which is now obsolete. Use Functions 4300h and 4310h (Chapter 38) instead.

**See also:** Functions 160Ah, 1610h, and 4680h, XMS Function 4300h (Chapter 38)

---

### INT 2Fh Function 1602h

*MS Windows/386 2.x*

***GET API ENTRY POINT***

**Registers at call:**
AX = 1602h

**Return Registers:**
ES:DI -> Windows/386 2.x API procedure entry point

**Details:** This interface is supported in Windows 3.x only for 2.x compatibility.

To get the current virtual machine (VM) ID in Windows/386 2.x:
AX = 0000h
ES:DI -> return address
JUMP to address returned from INT 2Fh Function 1602h
After JUMP, at return address:
BX = current VM ID.

**See also:** AfterDark Function C020h (Chapter 14)

---

### INT 2Fh Function 1603h
### *GET INSTANCE DATA*

*MS Windows/386*

**Registers at call:**
AX = 1603h

**Return Registers:**
AX = 5248h ('RH') if supported
DS:SI -> Windows/386 instance data (see Table 43-1)

**Details:** Reportedly supported by RM Nimbus MS-DOS 3.3 kernel.

This function is called by DOSMGR when Function 1607h Subfunction 0015h is not supported, as is the case in DOS versions prior to 5.0.

See Geoff Chappell's book *DOS Internals* for additional discussions of this function, DOSMGR's behavior, and instancing in general.

**See also:** Function 1607h Subfunction 0015h

**43-1 Format of Windows/386 instance data:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of IO.SYS (0000h = default 0070h) |
| 02h | WORD | offset in IO.SYS of STACKS data structure (DOS 3.2x)<br>0000h if not applicable |
| 04h | WORD | number of instance data entries (max 32) |
| 06h | Array of instance data entries | |
| 00h | WORD | segment (0002h = DOS kernel) |
| 02h | WORD | offset |
| 04h | WORD | size |

---

### INT 2Fh Function 1605h
### *WINDOWS ENHANCED MODE & 286 DOSX INIT BROADCAST*

*MS Windows*
*Callout*

**Registers at call:**
AX = 1605h
ES:BX = 0000h:0000h
DS:SI = 0000h:0000h
CX = 0000h
DX = flags
bit 0 = 0 if Windows enhanced-mode initialization
bit 0 = 1 if Microsoft 286 DOS extender initialization
bits 1-15 reserved (undefined)
DI = version number (major in upper byte, minor in lower)

**Return Registers:**
CX = 0000h if okay for Windows to load
CX nonzero if Windows should not load
ES:BX -> startup info structure (see Table 43-2)
DS:SI -> virtual86 mode enable/disable callback or 0000h:0000h

**Details:** The Windows enhanced mode loader and Microsoft 286 DOS extender will broadcast an INT 2Fh Function 1605h call when initializing. Any DOS device driver or TSR can watch for this broadcast and return the appropriate values. If the driver or TSR returns CX nonzero, it is also its responsibility to display an error message.

Each handler must first chain to the prior INT 2Fh handler with registers unchanged before processing the call. If the handler requires local data on a per-VM basis, it must store the returned ES:BX in the "next" field of a startup info structure and return a pointer to that structure in ES:BX.

A single TSR may set the V86 mode enable/disable callback; if DS:SI is already nonzero, the TSR must fail the initialization by setting CX nonzero.

**See also:** Functions 1606h and 1608h, Task Switcher Function 4B05h (Chapter 49)

### 43-2 Format of Startup Information Structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | major, minor version of info structure |
| 02h | DWORD | pointer to next startup info structure or 0000h:0000h |
| 06h | DWORD | pointer to ASCIZ name of virtual device file or 0000h:0000h |
| 0Ah | DWORD | virtual device reference data (see Table 43-4) (only used if above nonzero) |
| 0Eh | DWORD | pointer to instance data records or 0000h:0000h |

### 43-3 Format of one Instance Item in array:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | address of instance data (end of array if 0000h:0000h) |
| 04h | WORD | size of instance data |

### 43-4 Format of Virtual Device Reference Data:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | physical address of *unknown item* or 00000000h |
| 04h | DWORD | physical address of *unknown* table |
| 08h | DWORD | "DEST_PAGE" address to which pages must be mapped |
| 0Ch | N DWORDs | "SRC_PAGE" physical addresses of the pages (00000000h = end of table) |

**Note:** EMM386.EXE sets the first pointer to the start of the device driver chain, the second pointer to a field of 40h bytes followed by a 16-bit offset to the end of the SRC_PAGE table, and DEST_PAGE to the start segment of the UMB area.

### 43-5 Virtual mode enable/disable procedure calling convention:

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 0000h disable V86 mode | CF set on error |
| AX = 0001h enable V86 mode | CF clear if successful |
| interrupts disabled | interrupts disabled |

---

## INT 2Fh Function 1606h

*MS Windows*
*Callout, Real Mode*

### *WINDOWS ENHANCED MODE & 286 DOSX EXIT BROADCAST*

**Registers at call:**
AX = 1606h
DX = flags
bit 0 = 0 if Windows enhanced-mode exit
bit 0 = 1 if Microsoft 286 DOS extender exit
bits 1-15 reserved (undefined)

**Details:** If the initialization broadcast fails (Function 1605h returned CX nonzero), then this broadcast will be issued immediately.

**See also:** Functions 1605h and 1609h

---

## INT 2Fh Function 1607h

*MS Windows*

### *VIRTUAL DEVICE CALL OUT API*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1607h | (usually) AX,BX,CX,DX,ES contain results |
| BX = virtual device ID (see Table 43-8) | |
| CX = (usually) callout subfunction | |

**Details:** More of a convention than an API, this call specifies a standard mechanism for Windows enhanced-mode virtual devices (VxD's) to talk to DOS device drivers and TSRs.

See below for details on several virtual devices.

**See also:** Function 1605h; Function 1607h Subfunctions 000Ch, 0014h, and 0015h; Function 1684h; AfterDark Function C020h (Chapter 14)

---

### INT 2Fh Function 1608h
### *WINDOWS ENHANCED MODE INIT COMPLETE BROADCAST*

*MS Windows Callout*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1608h | none |

**Details:** This function is called after all installable devices have been initialized.

Real-mode software may be called between the Windows enhanced-mode init call (Function 1605h) and this call; the software must detect this situation.

**See also:** Functions 1605h and 1609h

---

### INT 2Fh Function 1609h
### *WINDOWS ENHANCED MODE BEGIN EXIT BROADCAST*

*MS Windows Callout*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1609h | none |

**Details:** This function is called at the beginning of a normal exit sequence; it is not made in the event of a fatal system crash.

**See also:** Functions 1606h and 1608h

---

### INT 2Fh Function 160Ah
### *IDENTIFY WINDOWS VERSION AND TYPE*

*MS Windows 3.1*

| Registers at call: | Return Registers: |
|---|---|
| AX = 160Ah | AX = 0000h if call supported<br>BX = version (BH=major, BL=minor)<br>CX = mode (0002h = standard, 0003h = enhanced) |

**See also:** Functions 1600h and 4680h

---

### INT 2Fh Function 160Bh
### *IDENTIFY TSRs*

*MS Windows 3.1*

**Purpose:** This call allows Windows-aware TSRs to make themselves known to Windows.

| Registers at call: | Return Registers: |
|---|---|
| AX = 160Bh<br>ES:DI -> communication structure (see Table 43-6) or 0000h:0000h | ES:DI -> communication structure |

**Details:** The TSR should allocate a communication structure, place the given ES:DI pointer in the first field, and return a pointer to the new structure.

**See also:** Functions 1605h and 160Ch, Task Switcher Functions 4B01h and 4B05h (Chapter 49)

**43-6 Format of TSR-to-Windows communication structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to next structure |
| 04h | WORD | PSP segment |
| 06h | WORD | API version ID (0100h) |
| 08h | WORD | EXEC flags |
| | | bit 0: "WINEXEC" |
| | | bit 1: "LOADLIBRARY" |
| | | bit 2: "OPENDRIVER" |
| 0Ah | WORD | "exec_cmd_show" |
| 0Ch | DWORD | "exec_cmd" |
| 10h | 4 BYTEs | reserved (0) |
| 14h | DWORD | pointer to TSR ID block (see Table 43-7) |
| 18h | DWORD | pointer to TSR data block or 0000h:0000h |

**43-7 Format of Norton Utilities 6.0 TSR ID block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | length of name string |
| 02h | N BYTEs | name of TSR's executable |

**INT 2Fh Function 160Ch** *MS Windows 3.1*

***DETECT ROMs***

**Registers at call:**
AX = 160Ch
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** This function is used by ROM Windows.

**See also:** Function 160Bh

**INT 2Fh Function 1680h** *MS Windows, DPMI, various*

***RELEASE CURRENT VIRTUAL MACHINE TIME-SLICE***

**Registers at call:**
AX = 1680h

**Return Registers:**
AL = 00h if the call is supported
AL = 80h (unchanged) if the call is not supported

**Details:** Programs can use this function in idle loops to enhance performance under multitaskers; this call is supported by MS Windows 3.0, DOS 5+, DPMI 1.0+, and in OS/2 2.x for multitasking DOS applications.

This function does not block the program; it just gives up the remainder of the time slice.

This function should not be used by Windows-specific programs.

**See also:** TopView INT 15h Function 1000h (Chapter 44), APM INT 15h Function 5305h (Chapter 19), European MS-DOS INT 21h Function 89h (Chapter 25), Novell NetWare INT 7Ah Function 000Ah (NI)

**INT 2Fh Function 1681h** *MS Windows 3+*

***BEGIN CRITICAL SECTION***

**Registers at call:**
AX = 1681h

**Return Registers:**
none

**Details:** This function is used to prevent a task switch from occurring. It should be followed by an INT 2Fh Function 1682h call as soon as possible.

Nested calls are allowed, and must be followed by an appropriate number of "end critical section" calls.

This function is not supported in Windows/386 2.x. Get the InDOS flag with INT 21h Function 34h and increment by hand.

**See also:** Function 1682h, TopView INT 15h Function 101Bh (Chapter 44), MS-DOS INT 21h Function 34h (Chapter 24)

**INT 2Fh Function 1682h** *MS Windows 3+*

***END CRITICAL SECTION***

**Registers at call:**
AX = 1682h

**Return Registers:**
none

**Details:** This function is not supported in Windows/386 2.x. Get the InDOS flag with INT 21h Function 34h and decrement it by hand, taking care not to decrement the InDOS flag through zero.

**See also:** Function 1681h, TopView INT 15h Function 101Ch (Chapter 44), MS-DOS INT 21h Function 34h (Chapter 24)

**INT 2Fh Function 1683h** *MS Windows 3+*

***GET CURRENT VIRTUAL MACHINE ID***

**Registers at call:**
AX = 1683h

**Return Registers:**
BX = current virtual machine (VM) ID

**Details:** Windows itself currently runs in VM 1, but this can't be relied upon.
VM IDs are reused when VMs are destroyed.
An ID of 0 will never be returned.

**See also:** Functions 1684h, 1685h, and 168Bh

---

## INT 2Fh Function 1684h

*MS Windows*

### *GET DEVICE API ENTRY POINT*

**Registers at call:**
AX = 1684h
BX = virtual device (VxD) ID (see Table 43-8)
ES:DI = 0000h:0000h

**Return Registers:**
ES:DI -> VxD API entry point, or 0:0 if the VxD does not support an API

**Details:** Some Windows enhanced-mode virtual devices provide services that applications can access. For example, the Virtual Display Device (VDD) provides an API used in turn by WINOLDAP.

**See also:** Function 1683h

**43-8 Values for VxD ID:**

| *Value* | *Name* | *CallOut* | *V86* | *PM* | *Description* |
|---|---|---|---|---|---|
| 01h | VMM | | | | Virtual Machine Manager |
| 02h | Debug | | | | |
| 03h | VPICD | Y | Y | | Virtual Prog. Interrupt Controller (PIC) Device |
| 04h | VDMAD | N | N | | Virtual Direct Memory Access (DMA) Device |
| 05h | VTD | Y | Y | | Virtual Timer Device |
| 06h | V86MMGR | N | N | | Virtual 8086 Mode Device |
| 07h | PAGESWAP | N | N | | Paging Device |
| 08h | Parity | N | N | | Parity-check trapper |
| 09h | Reboot | N | Y | | Ctrl-Alt-Del handler |
| 0Ah | VDD | N | Y | | Virtual Display Device (GRABBER) |
| 0Bh | VSD | N | N | | Virtual Sound Device |
| 0Ch | VMD | Y | Y | Y | Virtual Mouse Device |
| 0Dh | VKD | N | Y | | Virtual Keyboard Device |
| 0Eh | VCD | N | Y | | Virtual COMM Device |
| 0Fh | VPD | | | | Virtual Printer Device |
| 10h | VHD | | | | Virtual Hard Disk Device (Windows 3.0) |
| 10h | BLOCKDEV | N | N | | Virtual Hard Disk Device (Windows 3.1) |
| 11h | VMCPD | | | | Virtual Math CoProcessor Device |
| 12h | EBIOS | N | N | | Reserve EBIOS page (e.g., on PS/2) |
| 13h | BIOSXLAT | | | | Map ROM BIOS API between prot & V86 mode |
| 14h | VNETBIOS | Y | N | N | Virtual NetBIOS Device |
| 15h | DOSMGR | Y | N | | |
| 16h | WINLOAD | | | | |
| 17h | SHELL | N | Y | | |
| 18h | VMPoll | N | N | | |
| 19h | VPROD | | | | |
| 1Ah | DOSNET | N | N | | assures network integrity across VMs |
| 1Bh | VFD | N | N | | Virtual Floppy Device |
| 1Ch | VDD2 | | | | Secondary display adapter |
| 1Dh | WINDEBUG | N | Y | | |
| 1Eh | TSRLoad | | | | TSR instance utility |
| 1Fh | BiosHook | | | | BIOS interrupt hooker VxD |
| 20h | Int13 | N | N | N | |

| | | | | |
|---|---|---|---|---|
| 21h | PageFile | N | Y | Paging File device |
| 22h | SCSI | | | |
| 23h | MCA_POS | | | |
| 24h | SCSIFD | | | SCSI FastDisk device |
| 25h | VPEND | | | Pen device |
| 26h | APM | | | Advanced Power Management |
| 442h | VTDAPI | | | |
| 444h | VADMAD | | | Autoinitialize DMA (Windows 3.0) |
| 1025h | MMD | | | |

**Note:** The high bit of the VxD ID is reserved for future use. The next 10 bits are the OEM # which is assigned by Microsoft. The low 5 bits are the device number. Naturally, this scheme has not been adhered to.

### 43-9 VTD.386 calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 0000h get VTD version number | AH = major version<br>AL = minor version |
| AX = 0100h get current clock tick time | EDX:EAX = clock tick time in 840ns units since Windows was started |
| AX = 0101h get current system time in milliseconds | EAX = time in milliseconds that Windows has been running |
| AX = 0102h get current virtual machine time | EAX = cumulative amount of time the virtual machine has been active, in milliseconds |

**Details:** This API should only be called directly when TOOLHELP.DLL TimerCount() cannot be called .

### 43-10 VADMAD calling convention:

**Registers at call:**
DX = operation
    0000h set VADMAD mode
        AX = desired mode
    0001h set VADMAD channel
        AX = desired channel

**Return Registers:**
none

**Details:** After setting mode/channel, start the DMA operation with an OUT to I/O port 0Bh (channels 0-3) or D6h (channels 4-7).

---

## INT 2Fh Function 1685h

*MS Windows*

### *SWITCH VMs AND CALLBACK*

**Registers at call:**
AX = 1685h
BX = VM ID of virtual machine to switch to
CX = flags
    bit 0 wait until interrupts enabled
    bit 1 wait until critical section unowned
    bits 2-15 reserved (zero)
DX:SI = priority boost (see VMM.INC)
ES:DI -> FAR procedure to callback

**Return Registers:**
CF set on error
    AX = error code
        01h invalid VM ID
        02h invalid priority boost
        03h invalid flags
CF clear if successful
    event will be or has been called

**Details:** Some DOS devices, such as networks, need to call functions in a specific VM. This call forces the appropriate VM to be installed.

The callback procedure must preserve all registers and return with IRET.

**See also:** Function 1683h, DESQview INT 15h Function 1117h (Chapter 44)

---

### INT 2Fh Function 1688h
### *GET SELECTOR TO LDT*

*MS Windows 3.0, 386MAX v6.01 (undocumented)*

**Registers at call:**
AX = 1688h

**Return Registers:**
AX = *status* (0000h for 386MAX)
BX = selector for *unknown*

---

### INT 2Fh Function 1689h
### *KERNEL IDLE CALL*

*MS Windows 3.0+ (undocumented)*

**Registers at call:**
AX = 1689h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 1680h, TopView INT 15h Function 1000h (Chapter 44), MS-DOS INT 28h (Chapter 24)

---

### INT 2Fh Function 168Bh
### *SET FOCUS TO SPECIFIED VIRTUAL MACHINE*

*MS Windows 3.1*

**Registers at call:**
AX = 168Bh
BX = virtual machine ID (see Function 1683h)

**Return Registers:**
AL = 00h if focus set to specified VM

**Details:** This function is documented on the Microsoft Developer's Network CD-ROM.

**See also:** Function 1683h

---

### INT 2Fh Function 168Ch
### *RESTART COMMAND*

*MS Windows 3.1*

**Registers at call:**
AX = 168Ch
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** WIN.COM executes the specified application.

---

### INT 2Fh Function 4000h
### *GET VIRTUAL DEVICE DRIVER (VDD) CAPABILITIES*

*MS Windows 3.x (OS/2 2.x)*

**Registers at call:**
AX = 4000h

**Return Registers:**
AL = capabilities
- 01h does not virtualize video access
- 02h virtualizes the video when in text mode
- 03h virtualizes the video when in text mode or single plane graphics modes
- 04h virtualizes the video when in text mode, single plane graphics modes, and VGA multiplane modes
- FFh virtualizes the video fully

**Details:** Display drivers use this function to find out what capabilities exist for the VDD driver and also to trigger the VDD driver to call Functions 4005h and 4006h. This function also gives the Video Driver hardware access to the video registers.

---

### INT 2Fh Function 4680h
### *INSTALLATION CHECK*

*MS Windows v3.0 (undocumented)*

**Registers at call:**
AX = 4680h

**Return Registers:**
AX = status
- 0000h: MS Windows 3.0 running in real (/R) or standard (/S) mode, or DOS 5 DOSSHELL active
- nonzero: no Windows, Windows prior to 3.0, or Windows3 in enhanced mode

**Details:** Windows 3.1 finally provides an installation check which works in all modes (see Function 160Ah).
**See also:** Functions 1600h and 160Ah

---

### INT 2Fh Function 4B20h
*MS Windows 3*

### *WIN.COM - SET PROGRAM TO EXECUTE ON EXIT*

**Registers at call:**
AX = 4B20h

**Return Registers:**
AX = 0000h if successful
DX:CX -> 256-byte buffer for pathname and commandline (see Table 43-11)

**Details:** When the Windows function ExitWindows is called with an exit code of 44h, WIN.COM executes the program specified in the returned buffer and then restarts Windows.

**43-11 Format of buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 128 BYTEs | ASCIZ pathname of program to execute |
| 80h | 128 BYTEs | commandline for program |

---

### INT 6Fh Function 0204h
*MS Windows 3.0*
*Callout*

### *DOS APPLICATION SWITCH HOOK*

**Registers at call:**
AX = 0204h

**Return Registers:**
AX = status
0000h switch is allowed
other switch not allowed

**Details:** Intercepting this call will allow a DOS application to ensure that Windows will not switch away from it.
**Conflicts:** Novell NetWare (NI), 10Net (NI)

## Virtual Device Drivers

Windows uses "virtual devices" to bridge between the logical universe in which it runs and the physical universe which contains its hardware. This approach makes it possible to respond to interrupts and hardware actions without leaving protected mode, thus speeding up system operation. Communication between virtual devices and Windows itself is by means of interrupt APIs described in this section.

---

### INT 2Fh Function 1607h, Subfunction 0006h
*MS Windows*

### *"V86MMGR" VIRTUAL DEVICE API*

**Registers at call:**
AX = 1607h
BX = 0006h (VxD identifier of "V86MMGR")
CX = 0000h

**Return Registers:**
AX = status
0000h if local A20 state changed
1607h if A20 unchanged
other if global A20 state changed

---

### INT 2Fh Function 1607h, Subfunction 000Ch
*MS Windows*

### *"VMD" VIRTUAL DEVICE API*

**Registers at call:**
AX = 1607h
BX = 000Ch (VxD identifier of "VMD")

**Return Registers:**
CX = nonzero if mouse driver already virtualized

**Details:** VMD (Virtual Mouse Driver) calls this function and then checks whether CX is nonzero; if yes, it will not automatically virtualize the mouse driver. This would be used if MOUSE.COM already virtualizes itself using the Windows API.
**See also:** Function 1607h Subfunctions 0014h and 0015h

### INT 2Fh Function 1607h, Subfunction 0014h *MS Windows*
### *"VNETBIOS" VIRTUAL DEVICE API*

**Registers at call:**
AX = 1607h
BX = 0014H (VxD identifier of "VNETBIOS")

**Return Registers:**
ES:DI -> 128-byte table specifying VNETBIOS actions for each NetBIOS command code (see Table 43-12)

**Details:** VNETBIOS (Virtual NetBIOS) calls this function to determine whether the NetBIOS has an extension Windows should know about.

**See also:** Function 1607h Subfunction 000Ch and 0015h

43-12 Values for action code:

| | | |
|---|---|---|
| 00h | "VN_Unknown" | unknown command |
| 04h | "VN_No_Map" | no memory mapping necessary |
| 08h | "VN_Map_In" | input buffer is quickly used, so no global mapping needed |
| 0Ch | "VN_Map_In" | output buffer is quickly used, so no global mapping needed |
| 10h | "VN_Map_In_Out" | buffer is quickly used, so no global mapping needed |
| 14h | "VN_Chain_Send" | the chain-send command |
| 18h | "VN_Cancel" | special case for cancel command |
| 1Ch | "VN_Buffer_In" | buffer is incoming |
| 20h | "VN_Buffer_Out" | buffer is outgoing |
| 24h | "VN_Buffer_In_Out" | buffer used for both incoming and outgoing data |

### INT 2Fh Function 1607h, Subfunction 0015h *MS Windows*
### *"DOSMGR" VIRTUAL DEVICE API*

**Registers at call:**
AX = 1607h
BX = 0015h (VxD identifier of "DOSMGR")
CX = function
- 0000h query instance processing
  - DX = 0000h
- 0001h set patches in DOS
  - DX = bit mask of patch requests
    - bit 0 enable critical sections
    - bit 1 NOP setting/checking user ID
    - bit 2 turn INT 21h Function 3Fh on STDIN into polling loop
    - bit 3 trap stack fault in "SYSINIT" to WIN386
    - bit 4 BIOS patch to trap "Insert disk X:" to WIN386
- 0002h remove patches in DOS (ignored by DOS 5.0 kernel)
  - DX = bit mask of patch requests (see function 0001h)
- 0003h get size of DOS data structures
  - DX = bit mask of request (only one bit can be set)
    - bit 0: Current Directory Structure size
- 0004h determine instanced data structures

| | **Return Registers:** |
|---|---|
| | CX = state<br>0000h not instanced<br>other instanced (DOS 5+ kernel returns 0001h) |
| 0005h get device driver size<br>ES = segment of device driver | DX = segment of DOS drivers or 0000h for default of 0070h<br>ES:BX -> patch table (see Table 43-13) |
| | AX = B97Ch<br>BX = bit mask of patches applied<br>DX = A2ABh |
| | if supported request:<br>AX = B97Ch<br>CX = size in bytes of requested structure<br>DX = A2ABh<br>else:<br>all registers preserved |
| | AX = B97Ch if supported<br>DX = A2ABh if supported (DOS 5+ kernel returns 0000h)<br>BX = bit mask of instanced items<br>bit 0: CDS<br>bit 1: SFT<br>bit 2: device list<br>bit 3: DOS swappable data area |
| | DX:AX = 0000h:0000h on error (not dev. driver segment)<br>DX:AX = A2ABh:B97Ch if successful<br>BX:CX = size of device driver in bytes |

**Details:** DOSMGR (DOS Manager) will check whether the OEM DOS/BIOS data has been instanced via this API and will not perform its own default instancing of the normal DOS/BIOS data if so; if this API is not supported, DOSMGR will also try to access instancing data through INT 2Fh Function 1603h.

These functions are supported by the DOS 5+ kernel; DOSMGR contains tables of instancing information for earlier versions of DOS.

See Geoff Chappell's book _DOS_Internals_ for additional discussions of DOSMGR's behavior and instancing in general.

**See also:** Functions 1603h and 1605h, Function 1607h Subfunctions 000Ch and 0014h, Function 1684h

### 43-13 Format of patch table:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | DOS version (major, minor) |
| 02h | WORD | offset in DOS data segment of "SAVEDS" |
| 04h | WORD | offset in DOS data segment of "SAVEBX" |
| 06h | WORD | offset in DOS data segment of InDOS flag |
| 08h | WORD | offset in DOS data segment of User ID word |
| 0Ah | WORD | offset in DOS data segment of "CritPatch" table to enable critical section calls (see INT 2Ah Function 80h in *Network Interrupts*) |
| 0Ch | WORD | (DOS 5+ only) offset in DOS data segment of "UMB_HEAD", containing segment of last MCB in conventional memory |

---

### INT 66h

*MS Windows*

### *VITD.386 Virtual Interval Timer*

**Details:** This Windows 3.x Virtual Device Driver implements a virtual timer which will expire and call INT 66h. This timer can be used to calculate elapsed execution time etc.

**Conflicts:** see Table 2-3

## Video Support

Windows places stringent requirements upon system video and defines a set of API interrupts to handle them. Those interrupts are described in this section.

---

### INT 2Fh Function 4003h

*MS Windows 3.x*

### *ENTERING VIDEO DRIVER CRITICAL SECTION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4003h | none |

**Details:** This critical section must be exited within 1 second.

**See also:** Function 4004h

---

### INT 2Fh Function 4004h

*MS Windows 3.x*

### *EXITING VIDEO DRIVER CRITICAL SECTION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4004h | none |

**See also:** Function 4003h

---

### INT 2Fh Function 4005h

*MS Windows 3.x*
*Callout*

### *SWITCHING DOS TO BACKGROUND*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4005h | none |

**Details:** This function is called by Windows when the DOS box is about to be placed in the background and the video driver should save any necessary state information.

**See also:** Function 4006h, OS/2 Function 4001h (Chapter 26)

---

### INT 2Fh Function 4006h

*MS Windows 3.x*
*Callout*

### *SWITCHING DOS TO FOREGROUND*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4006h | none |

**Details:** This function is called by Windows when the DOS box is about to be placed in the foreground and the video driver should restore any necessary state information.

**See also:** Function 4005h, OS/2 Function 4002h (Chapter 26)

---

### INT 2Fh Function 4007h
*MS Windows 3.x*

### *ENABLE VDD TRAPPING OF VIDEO REGISTERS*

**Registers at call:**
AX = 4007h

**Return Registers:**
none

**Details:** This function is used by Windows Standard mode.

---

### INT 2Fh Function BC00h
*MS Windows 3.0, DOS 5+ EGA.SYS*

### *INSTALLATION CHECK*

**Registers at call:**
AX = BC00h

**Return Registers:**
AL = status
    00h not installed, OK to install
    01h not installed, not OK to install
    FFh installed
        BX = 5456h ("TV")

**Details:** AH=BCh is the default value, which may be changed by a command line parameter to any value between 80h and FFh.

**Conflicts:** MediaVision MVSOUND.SYS (Chapter 22); see also Table 2-2

**See also:** Function BC06h, Register Interface Library INT 10h Function FAh

---

### INT 2Fh Function BC06h
*MS Windows 3.0, DOS 5+ EGA.SYS (undocumented)*

### *GET VERSION INFO*

**Registers at call:**
AX = BC06h

**Return Registers:**
BX = 5456h ("TV")
CH = major version
CL = minor version
DL = revision

**Conflicts:** MediaVision MVSOUND.SYS (Chapter 22); see also Table 2-2

**See also:** Function BC00h, Register Interface Library INT 10h Function FAh (Chapter 8)

## WINOLDAP

WinOldAp (WINOLDAP.MOD) is a Microsoft Windows extension supporting "old" (character-mode) application access to Dynamic Data Exchange, menus, and the Windows clipboard.

---

### WINOLDAP INT 2Fh Function 1700h
*MS Windows*

### *IDENTIFY WinOldAp VERSION*

**Registers at call:**
AX = 1700h

**Return Registers:**
AX = 1700h if this version of WINOLDAP doesn't support clipboard
otherwise,
    AL = WINOLDAP major version
    AH = WINOLDAP minor version

**Details:** This installation check DOES NOT follow the format used by other software of returning AL=FFh when installed.

---

### WINOLDAP INT 2Fh Function 1701h
*MS Windows*

### *OPEN CLIPBOARD*

**Registers at call:**
AX = 1701h

**Return Registers:**
AX nonzero success
AX = 0000h clipboard is already open

## WINOLDAP INT 2Fh Function 1702h

*MS Windows*

### *EMPTY CLIPBOARD*

**Registers at call:**
AX = 1702h

**Return Registers:**
AX nonzero clipboard has been emptied
AX = 0000h failure

## WINOLDAP INT 2Fh Function 1703h

*MS Windows*

### *SET CLIPBOARD DATA*

**Registers at call:**
AX = 1703h
DX = clipboard format supported by WinOldAp (see Table 43-14)
ES:BX -> data (see Tables 43-15 and 43-16)
SI:CX = size of data

**Return Registers:**
AX nonzero data copied into the Clipboard
AX = 0 failure

**43-14 Clipboard formats:**

| | |
|---|---|
| 01h | text |
| 02h | bitmap |
| 03h | metafile picture |
| 04h | SYLK |
| 05h | DIF |
| 06h | TIFF |
| 07h | OEM text |
| 81h | DSP text |
| 82h | DSP bitmap |

**43-15 Format of bitmap:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | type (0000h) |
| 02h | WORD | width of bitmap in pixels |
| 04h | WORD | height of bitmap in pixels |
| 06h | WORD | bytes per line |
| 08h | BYTE | number of color planes |
| 09h | BYTE | number of adjacent color bits in pixel |
| 0Ah | DWORD | pointer to start of data |
| 0Eh | WORD | width in 0.1mm units |
| 10h | WORD | height in 0.1mm units |
| 12h | N BYTEs | bitmap data |

**43-16 Format of metafile picture:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | mapping mode |
| 02h | WORD | X extent |
| 04h | WORD | Y extent |
| 06h | WORD | picture data |

## WINOLDAP INT 2Fh Function 1704h

*MS Windows*

### *GET CLIPBOARD DATA SIZE*

**Registers at call:**
AX = 1704h
DX = clipboard format supported by WinOldAp (see Table 43-14)

**Return Registers:**
DX:AX = size of data in bytes, including any headers
DX:AX = 0000h if no data in this format in the Clipboard

---

## WINOLDAP INT 2Fh Function 1705h

*MS Windows*

### *GET CLIPBOARD DATA*

**Registers at call:**
AX = 1705h
DX = clipboard format supported by WinOldAp (see Table 43-14)
ES:BX -> buffer

**Return Registers:**
AX nonzero success
AX = 0000h error or no data in this format in Clipboard

---

## WINOLDAP INT 2Fh Function 1708h

*MS Windows*

### *CLOSE CLIPBOARD*

**Registers at call:**
AX = 1708h

**Return Registers:**
AX nonzero success
AX = 0000h failure

---

## WINOLDAP INT 2Fh Function 1709h

*MS Windows*

### *COMPACT CLIPBOARD*

**Registers at call:**
AX = 1709h
SI:CX = desired size in bytes

**Return Registers:**
DX:AX = number of bytes in largest block of free memory

**Details:** WinOldAp is responsible for including the size of any headers.

---

## WINOLDAP INT 2Fh Function 170Ah

*MS Windows*

### *GET DEVICE CAPABILITIES*

**Registers at call:**
AX = 170Ah
DX = GDI information index (see Table 43-17)

**Return Registers:**
AX = integer value of the desired item (see Tables 43-18 to 43-24)

**Details:** This function returns the device-capability bits for the given display.

**43-17 GDI Information indices:**

| | |
|---|---|
| 00h | device driver version |
| 02h | device classification |
| 04h | width in mm |
| 06h | height in mm |
| 08h | width in pixels |
| 0Ah | height in pixels |
| 0Ch | bits per pixel |
| 0Eh | number of bit planes |
| 10h | number of brushes supported by device |
| 12h | number of pens supported by device |
| 14h | number of markers supported by device |
| 16h | number of fonts supported by device |
| 18h | number of colors |
| 1Ah | size required for device descriptor |
| 1Ch | curve capabilities |
| 1Eh | line capabilities |
| 20h | polygon capabilities |
| 22h | text capabilities |
| 24h | clipping capabilities |
| 26h | bitblt capabilities |
| 28h | X aspect |
| 2Ah | Y aspect |
| 2Ch | length of hypotenuse of aspect |

| | |
|---|---|
| 58h | logical pixels per inch of width |
| 5Ah | logical pixels per inch of height |

**43-18 Values for device classification:**

| | |
|---|---|
| 00h | vector plotter |
| 01h | raster display |
| 02h | raster printer |
| 03h | raster camera |
| 04h | character-stream, PLP |
| 05h | Metafile, VDM |
| 06h | display-file |

**43-19 Bitfields for curve capabilities:**

| | |
|---|---|
| bit 0 | circles |
| bit 1 | pie wedges |
| bit 2 | chord arcs |
| bit 3 | ellipses |
| bit 4 | wide lines |
| bit 5 | styled lines |
| bit 6 | wide styled lines |
| bit 7 | interiors |

**43-20 Bitfields for line capabilities:**

| | |
|---|---|
| bit 1 | polylines |
| bit 2 | markers |
| bit 3 | polymarkers |
| bit 4 | wide lines |
| bit 5 | styled lines |
| bit 6 | wide styled lines |
| bit 7 | interiors |

**43-21 Bitfields for polygon capabilities:**

| | |
|---|---|
| bit 0 | polygons |
| bit 1 | rectangles |
| bit 2 | trapezoids |
| bit 3 | scanlines |
| bit 4 | wide borders |
| bit 5 | styled borders |
| bit 6 | wide styled borders |
| bit 7 | interiors |

**43-22 Bitfields for text capabilities:**

| | |
|---|---|
| bit 0 | output precision character |
| bit 1 | output precision stroke |
| bit 2 | clippping precision stroke |
| bit 3 | 90-degree character rotation |
| bit 4 | arbitrary character rotation |
| bit 5 | independent X and Y scaling |
| bit 6 | double-size |
| bit 7 | integer scaling |
| bit 8 | continuous scaling |
| bit 9 | bold |
| bit 10 | italic |
| bit 11 | underline |
| bit 12 | strikeout |

## WINOLDAP INT 2Fh Function 170Ah

| | |
|---|---|
| bit 13 | raster fonts |
| bit 14 | vector fonts |
| bit 15 | reserved |

**43-23 Values for clipping capabilities:**

| | |
|---|---|
| 00h | none |
| 01h | clipping to rectangles |

**43-24 Bitfields for raster capabilities:**

| | |
|---|---|
| bit 0 | simple bitBLT |
| bit 1 | device requires banding support |
| bit 2 | device requires scaling support |
| bit 3 | supports >64K bitmap. |

## WINX

WINX is a DOS/Windows utilities by Al Williams which can be used to launch Windows applications from a DOS Box; it was published in "DOS and Windows Protected Mode-Programming with DOS Extenders" (Addison-Wesley) and should not be confused with the Windows driver of the same name which is part of the DESQview/X package.

### INT 2Fh Function F200h

*WINX*

***INSTALLATION CHECK***

**Registers at call:**
AX = F200h

**Return Registers:**
AX = 00FFh if installed

**Conflicts:** none, but see Table 2-2
**See also:** Functions F201h to F203h

### INT 2Fh Function F201h

*WINX*

***RETURN ADDRESS OF SERVER BUFFER***

**Registers at call:**
AX = F201h

**Return Registers:**
AX = status
  0000h if successful
    BX:CX = address of server buffer (see Table 43-25)
  FFFFh if WINX is busy processing a different request

**Conflicts:** none, but see Table 2-2

**43-25 Format of server buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | command/status<br>00h buffer available<br>01h buffer contains result<br>02h change directory<br>03h execute prograem<br>FFh terminate windows portion of WINX |
| 01h | ? BYTEs | command (03h) or directory (02h) |
| or | | |
| 01h | DWORD | result (01h) |

### INT 2Fh Function F202h

*WINX*

***SET SERVER'S WORKING DIRECTORY***

**Registers at call:**
AX = F202h
BX:CX -> directory

**Return Registers:**
AX = status

0000h if successful
FFFFh if WINX is busy processing a different request

**Conflicts:** none, but see Table 2-2
**See also:** Functions F200h and F203h

---

### INT 2Fh Function F203h
### *EXECUTE COMMAND*

*WINX*

**Registers at call:**
AX = F203h
BX:CX -> command

**Return Registers:**
AX = status
0000h if successful
FFFFh if WINX is busy processing a different request

**Conflicts:** none, but see Table 2-2
**See also:** Functions F200h and F202h

# CHAPTER 44

# *TopView, DESQView, and DESQView/X*

Although IBM's TopView itself was short-lived, part or all of its API has been adopted by a number of other multitaskers, most notably Quarterdeck's DESQview. This chapter is divided into five sections: TopView calls, DESQview calls, DESQview/X calls, the DESQview External Device Interface, and a few miscellaneous calls. In addition, QEMM-386 (Chapter 39) provides the hooks which turn DESQview into DESQview/386, and other Quarterdeck utilities described in Chapter 63 may be of interest.

## TopView

IBM's TopView was one of the first multitaskers available for the MS-DOS environment, and was widely regarded as bloated, slow, and not particularly compatible. Its API, however, has far outlived TopView itself in the form of Quarterdeck's DESQview. Except as noted, all of the calls in this section are also available under DESQview. The function names listed here reflect the names used by DESQview; the TopView nomenclature may differ.

Though not specifically stated here, many of the TopView functions are also available under TaskView and its follow-on OmniView by Sunny Hill Software. See Chapter 49 for additional calls that are unique to OmniView.

All of the INT 15h Function 10XXh, 11XXh, and 12h functions in this section conflict with MultiDOS Plus (Chapter 45) and VMiX (Chapter 46).

---

### INT 10h Function FEh *TopView*
### *GET SHADOW BUFFER*

**Purpose:** Determine the address of the virtual screen to which the program should write instead of the actual video memory; this permits programs to be multitasked without interfering with each other's output, and allows memory managers to move the video memory to permit larger programs to be loaded.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = FEh | ES:DI -> actual video buffer for calling process |
| ES:DI -> assumed video buffer<br>B800h:0000h color text/CGA graphics,<br>B000h:0000h mono text, or<br>A000h:0000h EGA/VGA graphics (RSIS only) | |

**Details:** If no multitasker or Relocated Screen Interface Specification (RSIS)-compliant environment is installed, ES:DI is returned unchanged.

For display pages other than 0, use Functions 05h and 0Fh (Chapter 8) to determine whether a particular page exists.

TopView requires a call to Function FFh to notify it that the screen has changed; DESQview will check for changes itself until the first call to Function FFh.

**See also:** Function FFh, INT 15h Function 1024h, DESQview INT 21h Function 2Bh (below), Standard BIOS Function 05h (Chapter 8), AX PC Function 5201h (Chapter 6)DoubleDOS INT 21h Function ECh (Chapter 49)

---

### INT 10h Function FFh

*TopView*

### *UPDATE SCREEN FROM SHADOW BUFFER*

**Registers at call:**
AH = FFh
CX = number of consecutive changed characters
ES:DI -> first changed character in shadow buffer

**Return Registers:**
none

**Details:** Avoid CX=0000h, as some versions of DESQview hang the calling process.

DESQview will discontinue the automatic screen updating initiated by Function FEh after this call.

This function is not supported (it is ignored) by DESQview/X 1.0x.

**See also:** Function FEh, Alloy MW386 Function 93h (NI)

---

### INT 15h Function 1000h

*TopView*

### *"PAUSE" - GIVE UP CPU TIME*

**Registers at call:**
AX = 1000h

**Return Registers:**
after other processes run

**Details:** Under DESQview, if the process issuing this call has hooked INT 08h, the current time-slice is set to expire at the next clock tick rather than immediately.

**Conflicts:**

**See also:** MultiDOS Function 00h (Chapter 45), APM Function 5305h (Chapter 19), European MS-DOS INT 21h Function 89h (Chapter 25), DoubleDOS INT 21h Function EEh (Chapter 49), MS Windows INT 2Fh Function 1680h (Chapter 43), HP 95LX INT 60h Function 0106h (Chapter 20), Cswitch INT 62h Function 01h (Chapter 49), NetWare INT 7Ah Function 000Ah (NI), Alloy MW386 INT 7Fh Function E8h (NI)

---

### INT 15h Function 1001h

*TopView*

### *"GETMEM" - ALLOCATE "SYSTEM" MEMORY*

**Registers at call:**
AX = 1001h
BX = number of bytes to allocate

**Return Registers:**
ES:DI -> block of memory or 0000h:0000h (DV v2.26+)
AX = status (DV v2.42)
  0000h successful
  0001h failed

**Details:** Use SETERROR (Function DE15h) to avoid a user prompt if there is insufficient common memory. Under DV v2.42, this call never generates a user prompt regardless of the SETERROR value; instead, it always returns AX=0001h and ES:DI=0000h:0000h if out of memory.

**See also:** Functions 1002h, 102Eh, DE0Ch, and DE15h

---

### INT 15h Function 1002h

*TopView*

### *"PUTMEM" - DEALLOCATE "SYSTEM" MEMORY*

**Registers at call:**
AX = 1002h
ES:DI -> previously allocated block

**Return Registers:**
none

**See also:** Functions 1001h and DE0Dh

### INT 15h Function 1003h

*TopView*

***"PRINTC" - DISPLAY CHARACTER/ATTRIBUTE ON SCREEN***

**Registers at call:**
AX = 1003h
BH = attribute
BL = character
DX = segment of object handle for window

**Return Registers:**
none

**Details:** BX=0000h does not display anything, it only positions the hardware cursor.

### INT 15h Function 10h

*TopView*

***UNIMPLEMENTED IN DV 2.x***

**Registers at call:**
AH = 10h
AL = 04h thru 12h

**Return Registers:**
pops up "Programming error" window in DV 2.x

### INT 15h Function 1013h

*TopView*

***"GETBIT" - DEFINE A 2ND-LEVEL INTERRUPT HANDLER***

**Registers at call:**
AX = 1013h
ES:DI -> FAR service routine

**Return Registers:**
BX = bit mask indicating which bit was allocated
0000h if no more bits available

**See also:** Functions 1014h and 1015h

**Details:** Only a few TopView/DESQview API calls are allowed during a hardware interrupt; if other calls need to be made, the interrupt handler must schedule a 2nd-level interrupt with "SETBIT" (Function 1015h).

### INT 15h Function 1014h

*TopView*

***"FREEBIT" - UNDEFINE A 2ND-LEVEL INTERRUPT HANDLER***

**Registers at call:**
AX = 1014h
BX = bit mask from Function 1013h

**Return Registers:**
none

**See also:** Functions 1013h and 1015h

### INT 15h Function 1015h

*TopView*

***"SETBIT" - SCHEDULE ONE OR MORE 2ND-LEVEL INTERRUPTS***

**Registers at call:**
AX = 1015h
BX = bit mask for interrupts to post

**Return Registers:**
indicated routines will be called:
(DV 2.0x) at next task switch
(DV 2.2x) immediately on return from hardware interrupt

**See also:** Functions 1013h and 1014h

**Details:** This is one of the few TopView calls which are allowed from a hardware interrupt handler.

The second-level handler will be called with ES containing the segment of the handle of the next task to be executed; on return, ES must be the segment of a task handle. By modifying ES, the handler can force a particular task to be executed immediately.

### INT 15h Function 1016h

*TopView*

***"ISOBJ" - VERIFY OBJECT HANDLE***

**Registers at call:**
AX = 1016h
ES:DI = possible object handle

**Return Registers:**
BX = result
FFFFh valid object handle
0000h not a valid handle

**Details:** Under DESQview versions prior to 2.50, an object handle is always a pointer to the object; for versions 2.50 and up, only task handles are always pointers (other handles may consist of a unique object number and an offset into DESQview's common memory).
**See also:** Functions DE14h, DE2Bh, and DE2Ch

### 44-1 Format of object:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | offset in common memory of previous object of same type |
| 02h | WORD | offset in common memory of next object of same type |
| 04h | WORD | signature FEDCh (DV 2.42-)<br>signature FEDCh or object number (DV 2.50+) |
| 06h | WORD | object type<br>00h window/task, 01h mailbox, 02h keyboard, 03h timer, 04h pointer, 05h panel, 06h objectq |
| 08h | DWORD | object handle to return to caller |
| 0Ch | DWORD | canonicalized object address (segment = common memory) |
| 10h | WORD | offset in common memory of owning task |
| 12h | WORD | mapping context<br>offset in common memory of mapping context record (see Table 44-2)<br>remainder varies by object type and DESQview version |

### 44-2 Format of mapping context record:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of start of system memory chain |
| 02h | WORD | size of process's memory in paragraphs |
| 04h | 15 BYTEs | *unknown* |
| 13h | BYTE | reference count |

### INT 15h Function 1017h
*TopView*

### *UNIMPLEMENTED IN DV 2.x*

**Registers at call:**
AX = 1017h

**Return Registers:**
pops up "Programming error" window in DV 2.x

### INT 15h Function 1018h
*TopView*

### *"LOCATE" - FIND WINDOW AT A GIVEN SCREEN LOCATION*

**Registers at call:**
AX = 1018h
BH = column
BL = row
ES = segment of object handle for window below which to search
0000h = start search with topmost window

**Return Registers:**
ES = segment of object handle for window which is visible at the indicated position, or covered by indicated window; 0000h if no window

**See also:** Functions 1023h and 1024h

### INT 15h Function 1019h
*TopView*

### *"SOUND" - MAKE TONE*

**Registers at call:**
AX = 1019h
BX = frequency in Hertz (0000h = silence)
CX = duration in clock ticks (18.2 ticks/sec)

**Return Registers:**
immediately, tone continues to completion

**Details:** If another tone is already playing, the new tone does not start until completion of the previous one. Up to 32 tones may be queued before the process is blocked until a note completes.

In DV 2.00, the lowest tone allowed is 20 Hz.

If CX = 0000h, the current note is cancelled; if BX = 0000h as well, all queued notes are cancelled.

**See also:** FAKEY INT 16h Function 73h (Chapter 13)

---

### INT 15h Function 101Ah *TopView*

### *"OSTACK" - SWITCH TO TASK'S INTERNAL STACK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 101Ah | stack switched |

**Details:** This call may not be nested; a second call must be preceded by a call to "USTACK" (Function 1025h).

While TopView requires many API calls to be executed while on the task's internal stack, DESQview allows those calls to be executed regardless of the current stack.

**See also:** Function 1025h

### INT 15h Function 101Bh *TopView*

### *"BEGINC" - BEGIN CRITICAL REGION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 101Bh | task-switching temporarily disabled |

**Details:** TopView/DESQview will not task-switch until "ENDC" (Function 101Ch) is called unless the task voluntarily releases the CPU (upon regaining the CPU, task-switching will again be disabled).

Under DESQview, this function first suspends the caller until DOS is free.

**See also:** Functions 101Ch, DE13h, and DE1Ch; MultiDOS Function 0Dh (Chapter 45); MS Windows INT 2Fh Function 1681h (Chapter 43); HP 95LX INT 60h Function 0602h (Chapter 20)

### INT 15h Function 101Ch *TopView*

### *"ENDC" - END CRITICAL REGION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 101Ch | task-switching enabled |

**Details:** This API call may be made from within a hardware interrupt handler.

**See also:** Functions 101Bh, DE13h, and DE1Bh; MS Windows INT 2Fh Function 1682h (Chapter 43); HP 95LX INT 60h Function 0603h (Chapter 20)

### INT 15h Function 101Dh *TopView*

### *"STOP" - STOP TASK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 101Dh<br>ES = segment of object handle for task to be stopped (same as handle of main window for that task) | indicated task will not get any CPU time until restarted with Function 101Eh |

**Details:** Once a task has been stopped, additional "STOP"s are ignored.

**BUG:** In DV 2.00, this function is ignored unless the indicated task is the current task.

**See also:** Functions 101Eh and 102Bh, VMiX Function 12h (Chapter 46), European MS-DOS INT 21h Function 81h (Chapter 25)

### INT 15h Function 101Eh *TopView*

### *"START" - START TASK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 101Eh<br>ES = segment of object handle for task to be started (same as handle of main window for that task) | indicated task is started up again |

**Details:** Once a task has been started, additional "START"s are ignored.

**See also:** Functions 101Dh and 102Bh, European MS-DOS INT 21h Function 82h (Chapter 25)

### INT 15h Function 101Fh

*TopView*

### *"DISPEROR" - POP-UP ERROR WINDOW*

**Registers at call:**
AX = 101Fh
BX = display options (see Table 44-3)
ES:DI -> text of message
CH = width of error window (0 = default)
CL = height of error window (0 = default)
DX = segment of object handle

**Return Registers:**
BX = reason for removal
  0001h left button pressed
  0002h right button pressed
  001Bh ESC pressed

**Details:** The popup window remains on-screen until ESC or the indicated mouse button is pressed.

**44-3 Bitfields for display options:**

| | |
|---|---|
| bits 0-12 | number of characters to display |
| bits 13,14 | which mouse button may be pressed to remove window<br>(00 = either, 01 = left, 10 = right, 11 = either) |
| bit 15 | beep if set |

### INT 15h Function 1020h

*TopView*

### *UNIMPLEMENTED IN DV v2.00+*

**Registers at call:**
AX = 1020h

**Return Registers:**
pops up "Programming error" window in DV v2.00+

### INT 15h Function 1021h

*TopView*

### *"PGMINT" - INTERRUPT ANOTHER TASK*

**Registers at call:**
AX = 1021h
BX = segment of object handle for task to interrupt (not self)
DX:CX -> FAR routine to invoke next time task is run
BP, SI, DI, DS, ES = values to be passed to task

**Return Registers:**
none

**Details:** The FAR routine is entered with the current ES, DS, SI, DI, and BP values, using the task's internal stack (see Function 101Ah); only SS:SP needs to be preserved.

Multiple PGMINTs to a single task are processed last-in first-out.

If the other task is in a DOS or DV API call, the interruption will occur on return from that call.

### INT 15h Function 1022h, Subfunction 0000h

*TopView*

### *"GETVER" - GET VERSION*

**Registers at call:**
AX = 1022h
BX = 0000h

**Return Registers:**
BX nonzero, TopView or compatible loaded
(BL = major version, BH = minor version)

**Details:** TaskView returns BX = 0001h, DESQview v2.00+ returns BX = 0A01h.

### INT 15h Function 1023h

*TopView*

### *"POSWIN" - POSITION WINDOW*

**Registers at call:**
AX = 1023h
BX = segment of object handle for parent
  window within which to position the window (0 = full screen)
ES = segment of object handle for window to be positioned
DL = positioning options (see Table 44-4)
CH = number of columns to offset from position specified by DL
CL = number of rows to offset from position specified by DL

**Return Registers:**
none

**44-4 Bitfields for positioning options:**

bits 0,1 horizontal position
- 00 = current
- 01 = center
- 10 = left
- 11 = right

bits 2,3 vertical position
- 00 = current
- 01 = center
- 10 = top
- 11 = bottom

bit 4 don't redraw screen if set

bits 5-7 not used

---

### INT 15h Function 1024h
*TopView*

### *"GETBUF" - GET VIRTUAL SCREEN INFO*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1024h | ES:DI -> virtual screen |
| BX = segment of object handle for window (0 = use default) | CX = size of virtual screen in bytes |
| | DL = screen type (00h text, 01h graphics) |

**See also:** INT 10h Function FEh above, DESQview INT 21h Function 2Bh Subfunction 4445h (below)

### INT 15h Function 1025h
*TopView*

### *"USTACK" - SWITCH BACK TO USER'S STACK*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1025h | stack switched back |

**Details:** Call this function only after having switched to the internal stack with Function 101Ah.

While TopView requires many API calls to be executed while on the task's private stack, DESQview allows those calls to be executed regardless of the current stack.

**See also:** Function 101Ah

### INT 15h Function 10h
*DESQview (TopView)*

### *UNIMPLEMENTED IN DV 2.x*

| Registers at call: | Return Registers: |
|---|---|
| AH = 10h | pops up "Programming error" window in DV 2.x |
| AL = 26h thru 2Ah | |

### INT 15h Function 102Bh
*DESQview v2.00+ (TopView)*

### *"POSTTASK" - AWAKEN TASK*

**Purpose:** Force a task which is waiting on its objectq to continue by placing the handle for the task on the objectq.

| Registers at call: | Return Registers: |
|---|---|
| AX = 102Bh | none |
| BX = segment of object handle for task | |

**See also:** Functions 101Dh and 101Eh, European MS-DOS INT 21h Function 82h (Chapter 25)

### INT 15h Functions 11XXh
*TopView commands*

| Registers at call: | Return Registers: |
|---|---|
| AH = 11h | vary by function |
| AL = various (except 17h) | |
| other registers as appropriate for function | |

**Details:** In DESQview 2.x, these function calls are identical to Functions DEXXh, so see those below.
**See also:** DESQview Functions DE00h to DE31h (below)

---

### INT 15h Function 12h, Subfunction 00h

*TopView*

### *SEND MESSAGE - "HANDLE" - RETURN OBJECT HANDLE*

**Registers at call:**
AH = 12h
BH = 00h
BL = handle identifier (see Table 44-5)

**Return Registers:**
STACK: DWORD object handle

**Details:** BL=0Ch/0Dh returns 00000000h if the object is not open (keyboard, mailbox, panel, pointer, and timer objects) or is an orphan (task, window).
**See also:** Function 12h Subfunctions 02h and 80h

**44-5 Handle identifiers:**

| | |
|---|---|
| 00h | handle in DWORD on top of stack |
| 01h | current task's window handle |
| 02h | given task's mailbox handle (task's handle on top of stack) |
| 03h | current task's mailbox handle |
| 04h | given task's keyboard handle (task's handle on top of stack) |
| 05h | current task's keyboard object handle |
| 06h | given task's OBJECTQ handle (task's handle on top of stack) |
| 07h | current task's OBJECTQ handle |
| 08h to 10h | return 00000000h under DV < 2.26 |
| 0Ch | (2.26+) task owning object with handle in DWORD on top of stack |
| 0Dh | (2.26+) task handle of owner (parent) of current task |

---

### INT 15h Function 12h, Subfunction 01h

*TopView*

### *SEND MESSAGE - "NEW" - CREATE NEW OBJECT*

**Registers at call:**
AH = 12h
BH = 01h
BL = object type to create (see Table 44-6)
STACK contains parameters if creating window (see Table 44-7)

**Return Registers:**
DWORD on top of stack is new object handle

**Details:** If a new task is created, it is started with AX = BX = SI = DI = BP = 0; DX:CX = handle of parent task; and DS = ES = SS = segment of private stack (and new task's handle).

New windows are orphans, inherit the colors/hidden status of the creating task's window, and are placed in the upper left hand corner of the screen but not automatically redrawn.

New keyboards are closed, and have all object bits cleared except for the hardware cursor bit.
**See also:** Function 12h Subfunctions 02h and 81h

**44-6 Object types for creation:**

| | |
|---|---|
| 00h | (DV 2.0x only) same type as handle in DWORD on top of stack |
| 01h | (DV 2.0x only) use task's window handle |
| 02h | (DV 2.0x only) given task's mailbox (task's handle on stack) |
| 03h | (DV 2.0x only) current task's mailbox |
| 04h | (DV 2.0x only) given task's keyboard (task's handle on stack) |
| 05h | (DV 2.0x only) current task's keyboard object |
| 08h | WINDOW class |
| 09h | MAILBOX class |
| 0Ah | KEYBOARD class |

0Bh TIMER object (counts down 32-bit time in 10ms increments)
0Fh POINTER object
10h PANEL object

**44-7 Window object creation parameters on stack:**

TOS:

| | |
|---|---|
| DWORD | address to jump to (no new task if high word == 0) |
| DWORD | (reserved) 0 = non-task window, FFFFh = task window |
| DWORD | bytes for task's private stack (FFFFh == default of 0100h) |
| DWORD | bytes system memory for input buffer for READ/READN<br>(0 == none, -1 == default--same as logical window size) |
| DWORD | window size, columns |
| DWORD | window size, rows |
| DWORD | length of window title |
| DWORD | address of window title |

---

## INT 15h Function 12h, Subfunction 02h

*TopView*

### *SEND MESSAGE - "FREE" - FREE AN OBJECT*

**Registers at call:**
AH = 12h
BH = 02h
BL = handle identifier (see Table 44-5)
    window: close window and free
    task's window: kills task, never returns
    timer: free timer
    panel: free panel object
    pointer: free pointer
    mailbox: free mailbox
    keyboard: free keyboard
STACK: DWORD object's handle (if BL even)

**Return Registers:**
STACK popped if parameter passed on stack

**Details:** When a window is freed, its keyboard and pointer objects are freed; task windows also free any mailbox, objectq, and panel objects held by the task and any child tasks.

If the keyboard being freed is the default keyboard for a task, this call is equivalent to CLOSE.

Panel and pointer objects are automatically closed if open.

**See also:** Function 12h Subfunctions 01h, 0Dh, and 82h

---

## INT 15h Function 12h, Subfunction 03h

*TopView*

### *SEND MESSAGE - "ADDR" - GET HANDLE OF MESSAGE SENDER*

**Registers at call:**
AH = 12h
BH = 03h
BL = object identifier
    00h mailbox handle in DWORD on top of stack
    02h sender of last message read from mailbox (task's handle on stack)
    03h sender of last message read from caller's default mailbox
STACK: DWORD object's handle (if BL even)

**Return Registers:**
STACK: DWORD task handle of message sender

**See also:** Function 12h Subfunctions 00h and 83h

---

## INT 15h Function 12h, Subfunction 0300h

*TopView*

### *SEND MESSAGE - "DIR" - GET PANEL FILE DIRECTORY*

**Registers at call:**
AH = 12h

BX = 0300h
STACK: DWORD handle of panel object

**Return Registers:**
STACK: DWORD length of directory (always multiple of 14 bytes)
DWORD address of directory

**Details:** A null string is returned if the object is not open.
**See also:** Function 12h Subfunctions 0400h (APPLY) and 83h

**44-8 Format of panel file:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | C0h C3h |
| 02h | BYTE | number of panels in file |
| 03h | 14N BYTEs | for each panel in file: |
| | 8 BYTEs | blank-padded panel name |
| | DWORD | panel offset in file |
| | WORD | panel length |
| | var | data for panels (each consists of one or more data streams)<br>first byte of each panel must be 1Bh, fifth byte must be E5h |

---

### INT 15h Function 12h, Subfunction 04h

*TopView*

### *SEND MESSAGE - "READ" - READ NEXT LOGICAL LINE OF WINDOW*

**Registers at call:**
AH = 12h
BH = 04h
BL = handle identifier for window (see Table 44-5)
STACK: DWORD window's handle (if BL even)

**Return Registers:**
STACK: DWORD number of bytes read
DWORD address of buffer

**Details:** Reading starts at the current logical cursor position; the cursor is updated to point at the character following the last one read.

If logical attributes are active, any translucent blanks (FFh) which are visible on screen are changed to the character which is seen through them (except under DESQview 1.00-1.10).

The string produced by the read is placed in an input buffer which may be reused by the next READ or READN of a window.

Window stream opcodes D8h and D9h determine whether the read returns characters or attributes.

**See also:** Function 12h Subfunctions 05h (WINDOW), 12h, and 84h

---

### INT 15h Function 12h, Subfunction 04h

*TopView*

### *SEND MESSAGE - "READ" - GET NEXT RECORD FROM OBJECT*

**Registers at call:**
AH = 12h
BH = 04h
BL = handle identifier (see Table 44-5)
- mailbox: wait for and get next message
- keyboard: wait for and get pointer to next input buffer
- objectq: wait for input from any object in ObjectQ
- pointer: wait for and get next message

STACK: DWORD handle (if BL even)

**Return Registers:**
STACK: (objectq) DWORD handle of object with input
(otherwise) DWORD number of bytes
DWORD address

**Details:** For a keyboard in keystroke mode, the input buffer is a single byte containing the character code as returned by the BIOS; the BIOS scan code is available via the STATUS call if the character is zero. For a keyboard in field mode, the input buffer format is determined by the field table header for the window to which the keyboard is attached.

Keyboard input buffers and mailbox message buffers may be invalidated by the next READ, ERASE, CLOSE, or FREE message to the same object.

**See also:** Function 12h Subfunctions 05h (OBJECT) and 84h

### INT 15h Function 12h, Subfunction 04h

**44-9 Format of pointer message:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | row |
| 02h | WORD | column |
| 04h | BYTE | status<br>bit 7: set when press/release mode active and button pressed<br>bit 6: set when press/release mode active and button released<br>bits 7-2: number of clicks-1 if multiple-click mode active<br>bits 1,0: button pressed (00=none,01=button1,10=button2) |
| 05h | BYTE | field number or zero (APILEVEL >= 2.00 only) |

### INT 15h Function 12h, Subfunction 0400h *TopView*

### *SEND MESSAGE - "READ" - WAIT FOR TIMER TO EXPIRE*

**Registers at call:**
AH = 12h
BX = 0400h
STACK: DWORD timer's handle

**Return Registers:**
after timer expires
STACK: DWORD time in 1/100 sec after midnight when timer expired

**See also:** Function 12h Subfunctions 0Ah and 84h

### INT 15h Function 12h, Subfunction 0400h *TopView*

### *SEND MESSAGE - "APPLY" - WRITE PANEL TO WINDOW*

**Registers at call:**
AH = 12h
BX = 0400h
STACK: DWORD handle of panel object
DWORD window's handle (0 = current task's window)
DWORD length of panel name
DWORD pointer to panel name

**Return Registers:**
STACK: DWORD handle of window which was used
DWORD handle of keyboard or 00000000h

**Details:** The status of the APPLY may be checked with the STATUS message.

The panel **must** have the following format: the first byte must be 1Bh (i.e. must start with a stream), and the first opcode in stream must be E5h. The single byte arg of the opcode is interpreted thus:

bits 7,6 11 means create new window
10 means create new field table for existing window
01 means use existing window and field table
bit 5 if set, panel contains a field table (creates a new keyboard and puts it in field mode)
bit 4 if set, panel contains input fields
bit 3 if set, panel contains select fields but no input fields.

If the panel contains input or select fields, a keyboard handle is returned; either the window's current open keyboard or a newly-created keyboard object. The caller should read that keyboard to obtain input from the panel.

**See also:** Function 12h Subfunctions 84h

### INT 15h Function 12h, Subfunction 05h *TopView*

### *SEND MESSAGE - "WRITE" - WRITE TO OBJECT*

**Registers at call:**
AH = 12h
BH = 05h
BL = object identifier (see Table 44-10)
STACK contents as appropriate for object (see Table 44-11)

**Return Registers:**
STACK popped

**Details:** Under DV 2.2+, failed mailbox writes may return CF set (see Function DE15h).

The data and status written to a keyboard object must match the format returned by the keyboard object in the current mode.

The pointer position is scaled according to the current scaling factors.

**See also:** Function 12h Subfunctions 04h and 85h

### 44-10 Object identifiers:

| | |
|---|---|
| 00h | handle is DWORD on top of stack<br>timer: start timer to end at a specified time<br>keyboard: add input buffer to queue<br>pointer: move pointer icon to specified position |
| 02h | send message by value/status=0 to mbox (task's handle on stack) |
| 03h | send message by value/status=0 to current task's mailbox |
| 04h | add input buffer to KEYBOARD queue (handle on top of stack) |
| 05h | add input buffer to task's default KEYBOARD queue |
| 06h | add an object to OBJECTQ (handle on top of stack) |
| 07h | add an object to task's default OBJECTQ |

### 44-11 Write parameters on stack:

| | |
|---|---|
| (if mailbox) | DWORD length<br>DWORD address |
| (if keyboard) | DWORD status (scan code in keystroke mode)<br>DWORD length (should be 1 in keystroke mode)<br>DWORD address |
| (if objectq) | DWORD handle of object to add |
| (if timer) | DWORD 1/100ths seconds since midnight (actually only accurate to 1/18 sec) |
| (if pointer) | DWORD column relative to origin of window<br>DWORD row relative to origin of window |

## INT 15h Function 12h, Subfunction 05h

*TopView*

### *SEND MESSAGE - "WRITE" - WRITE STRING TO WINDOW*

**Registers at call:**

AH = 12h

BH = 05h

BL = window to which to write

- 00h DWORD on top of stack is window handle
- 01h write string to task's default window
- 0Ch (DV 2.26+) default window of task owning handle on top of stack
- 0Dh (DV 2.26+) default window of parent of current task

| STACK: | | |
|---|---|---|
| | DWORD | object handle if handle passed on stack |
| | DWORD | total length of string (high word == 0) |
| | DWORD | address of string to display |

**Return Registers:**

indicated actions performed

a. non-control characters are displayed (opcodes DEh and DFh control whether the attributes are left or changed to the current attrib)

b. CR/LF/BS/Tab cause the usual cursor movement

c. ESC starts a data structure with additional commands if following byte is less than 20h; otherwise, it is written to the window

STACK: DWORD handle of new window if window stream opcode E6h present else nothing

**See also:** Function 12h Subfunctions 04h and 85h

### 44-12 Data Structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | "magic" signature 1Bh |

| | | |
|---|---|---|
| 01h | BYTE | stream mode: 00h, 01h, 10h, 14h-1Fh |
| 02h | WORD | length of remainder of stream in bytes<br>variable-length fields follow, each an OPCODE followed by zero or more args |

---

### 44-13 MODE 00h "WINDOW STREAM" Opcodes:args

| | |
|---|---|
| 00h | display 20h blanks with the default attribute |
| 01h-1Fh | display OPCODE blanks with the default attribute |
| 20h | display char with default attribute 20h times<br>BYTE char to repeat |
| 21h-3Fh | display char with default attribute OPCODE-20h times<br>BYTE char to repeat |
| 40h | display 20h blanks with specified attribute<br>BYTE attribute of blanks |
| 41h-5Fh | display OPCODE-40h blanks with specified attribute<br>BYTE attribute of blanks |
| 60h | display next 20h characters<br>20h BYTEs characters to display |
| 61h-7Fh | display next OPCODE-60h characters<br>N BYTEs characters to display |
| 80h-87h | display N blanks with default attribute<br>BYTE low 8 bits of 11-bit count (high 3 in low 3 bits of OPCODE) [000h means 800h] |
| 88h-8Fh | display N copies of the character<br>BYTE low 8 bits of 11-bit count (high 3 in low 3 bits of OPCODE) [000h means 800h]<br>BYTE character to repeat |
| 90h-97h | display N blanks with specified attribute<br>BYTE low 8 bits of 11-bit length (high 3 in low 3 bits of OPCODE) [000h means 800h]<br>BYTE attribute |
| 98h-9Fh | display string at logical cursor pos<br>BYTE low 8 bits of 11-bit length (high 3 in low 3 bits of OPCODE) [000h means 800h]<br>N BYTEs string to display |
| A0h | set logical cursor row<br>BYTE row number (0 is top) |
| A1h | set logical cursor column<br>BYTE column number (0 is leftmost) |
| A2h | set top edge of scrolling region<br>BYTE row |
| A3h | set left edge of scrolling region<br>BYTE column |
| A4h | set row of physical window position<br>BYTE line |
| A5h | set column of physical window position<br>BYTE column |
| A6h | set height of physical window<br>BYTE #rows |
| A7h | set width of physical window<br>BYTE #columns |
| A8h | set viewport row<br>BYTE row |
| A9h | set viewport column |

| | |
|---|---|
| | BYTE column |
| AAh | set virtual screen height [contents of window unpredictable after] |
| | BYTE rows |
| ABh | set virtual screen width [contents of window unpredictable after] |
| | BYTE columns |
| ACh-AEh | unused |
| AFh | set compatible/preferred video modes |
| | BYTE compatibility/preference mask |
| | bit 7 compatible with monochrome |
| | bit 6 compatible with color text, EGA/VGA graphics |
| | bit 5 compatible with medium-resolution CGA graphics |
| | bit 4 compatible with high-resolution CGA graphics |
| | bit 3 prefer monochrome |
| | bit 2 prefer color text, EGA/VGA graphics |
| | bit 1 prefer medium-resolution CGA graphics |
| | bit 0 prefer high-resolution CGA graphics |
| B0h | move logical cursor down |
| | BYTE #rows (signed, negative values move up) |
| | [if #rows=0 and hardware cursor owner, update hw crsr] |
| B1h | move logical cursor right |
| | BYTE #cols (signed, negative values move left) |
| | [if #cols=0 and hardware cursor owner, update hw crsr] |
| B2h | shift top edge of scrolling region |
| | BYTE #rows (signed) |
| B3h | shift left edge of scrolling region |
| | BYTE #cols (signed) |
| B4h | shift physical window down |
| | BYTE #lines (signed) |
| B5h | shift physical window right |
| | BYTE #columns (signed) |
| B6h | expand physical window vertically |
| | BYTE #lines (signed) |
| B7h | expand physical window horizontally |
| | BYTE #columns (signed) |
| B8h | adjust viewport row |
| | BYTE #rows (signed) |
| B9h | adjust viewport column |
| | BYTE #columns (signed) |
| BAh | adjust virtual screen height [contents of window unpredict after] |
| | BYTE #rows to increase (signed) |
| BBh | adjust virtual screen width [contents of window unpredictbl after] |
| | BYTE #cols to increase (signed) |
| BCh-BFh | reserved (currently unused) |
| C0h | set logical cursor position |
| | BYTE row number (0 is top border) |
| | BYTE column number (0 is left border) |
| C1h | set top left corner of scrolling region |
| | BYTE row |
| | BYTE column |

| | |
|---|---|
| C2h | set physical window pos<br>BYTE upper left row (no top border if 0)<br>BYTE upper left column (no left border if 0) |
| C3h | set current window size<br>BYTE #rows<br>BYTE #cols |
| C4h | set upper left corner of viewport (portion of virtual screen displayed in window)<br>BYTE row<br>BYTE column |
| C5h | set size of virtual screen [contents unpredictable afterwards]<br>BYTE #rows<br>BYTE #cols |
| C6h | unused |
| C7h | unused |
| C8h | set logical cursor relative to current position<br>BYTE number of rows to move down (signed)<br>BYTE number of columns to move right (signed)<br>[if #rows=#cols=0 and hardware cursor owner, update hw cursr] |
| C9h | shift top left corner of scrolling region<br>BYTE #rows (signed)<br>BYTE #cols (signed) |
| CAh | set window pos relative to current position<br>BYTE number of rows to shift down (signed)<br>BYTE number of columns to shift right (signed) |
| CBh | set window size relative to current size<br>BYTE number of rows to expand (signed)<br>BYTE number of cols to expand (signed) |
| CCh | shift viewport relative to current position<br>BYTE rows to shift (signed)<br>BYTE cols to shift (signed) |
| CDh | resize virtual screen<br>BYTE #rows to expand (signed)<br>BYTE #cols to expand (signed) |
| CEh | scroll text when using E8h-EBh/F8h-FBh opcodes (default) |
| CFh | scroll attributes when using E8h-EBh/F8h-FBh opcodes |
| D0h | allow window frame to extend beyond screen |
| D1h | always display a complete frame, even if window extends beyond edge of screen |
| D2h | allow DV to change logical colors on video mode switch (default) |
| D3h | application changes logical attributes |
| D4h | window is visible [must redraw to actually make visible] |
| D5h | window is hidden [must redraw to actually remove] |
| D6h | window has frame (default) |
| D7h | window unframed [must redraw to actually remove frame] |
| D8h | READ/READN will read characters from window (default) |
| D9h | READ/READN will read attributes from window |
| DAh | use logical attributes, which may be remapped (see Table 44-17) |
| DBh | use physical attributes for characters |

DCh enable special actions for control characters (default)
DDh disable special control char handling, all characters displayable by
    BIOS TTY call
DEh write both character and attribute (default)
DFh write character only, leave attribute untouched
TopView;command repetition
E0h repeat following commands through E1h opcode
    BYTE number of times to repeat (00h means 256 times)
E1h end of commands to repeat, start repeating them
E2h set current output color
    BYTE color
E3h clear virtual screen from scroll origin to end using current color
E4h redraw window
TopView;menu style
E5h select menu style
    BYTE style (normally 18h)
        bits 5,4 = 01 use two-letter menu entries for remainder of
         this stream
E5h (panel file only)
    BYTE modifier
        bits 7,6 = 11 panel stream creates new window
                 = 10 panel defines new field table for existing window
                 = 01 panel stream uses existing window & field table
        bit 5 = 1 stream contains a field table (create kyboard object)
        bit 4 = 1 stream defines input fields (create keyboard object)
        bit 3 = 1 stream defines select fields but not input fields
        bit 2 = 1 stream defines exclusive input window (DV 2.2)
        bit 1 reserved
        bit 0 reserved
E6h create new window and perform rest of manipulations in new window
    BYTE number of rows
    BYTE number of columns

**Returns:** DWORD object handle of new window returned on stack at end

**Details:** The window is created with a physical size of 0x0 at the same position as the window to which this stream was sent.

E7h no operation
E8h scroll area up (top left corner defined by opcode C1h)
    BYTE height
    BYTE width
E9h scroll area down (top left corner defined by opcode C1h)
    BYTE height
    BYTE width
EAh scroll area left (top left corner defined by opcode C1h)
    BYTE height
    BYTE width
EBh scroll area right (top left corner defined by opcode C1h)
    BYTE height
    BYTE width
ECh set logical attributes for window contents

  - BYTE video modes command applies to
    - bit 7 monochrome
    - bit 6 color text, EGA/VGA graphics
    - bit 5 medium-resolution CGA graphics
    - bit 4 high-resolution CGA graphics
  - BYTE which attributes to set
    - bit 7 if set, copy single following byte to indicated attribs
    - bits 4-6 # of first attribute to change - 1
    - bits 0-3 # of consecutive attributes to change
  - N BYTEs new attributes
- EDh set logical attributes for window frame
  - BYTE video modes command applies to (see opcode ECh)
  - BYTE which attributes to set
    - bit 7 if set, copy single following byte to indicated attrs
    - bits 4-6 # of first attribute to change - 1
    - bits 0-3 # of consecutive attributes to change
  - N BYTEs new attributes
    - attributes
    - 1 = top left corner
    - 2 = top right corner
    - 3 = bottom left corner
    - 4 = bottom right corner
    - 5 = top edge
    - 6 = bottom edge
    - 7 = left edge
    - 8 = right edge
- EEh set characters for window frame
  - BYTE video modes command applies to (see opcode ECh)
  - BYTE which characters to set
    - bit 7 if set, copy single following byte to indicated characters
    - bits 4-6 # of first char to change - 1
    - bits 0-3 # of consecutive characters to change
  - N BYTEs new characters (same relative position as attributes above)
- EFh set window name
  - BYTE length of name (should be in range 0 to logical screen width)
  - N BYTEs name
- F0h clear input field to blanks
  - BYTE field number
- F1h fill input field with character
  - BYTE field number
  - BYTE char
- F2h set color of input field
  - BYTE field number (1-N)
  - BYTE attribute
- F3h set initial contents of input field
  - BYTE field number (1-N)
  - N BYTEs enough characters to exactly fill field as defined by opcode FFh
- F4h position cursor to start of specific input field
  - BYTE field number (1-N)

F5h change field table entry
BYTE field number
7-8 BYTEsfield table entry (see opcode FFh below)
F6h set field type
BYTE field number
BYTE type
00h inactive
40h output field
80h input field
C0h deselected field
C2h selected field
F7h "broadcast write" write data to fields with program output bit set in field table entry, in field number order
N BYTEs (total length of all program output fields)
F8h scroll field up a line
BYTE field number
F9h scroll field down a line
BYTE field number
FAh scroll field left
BYTE field number
FBh scroll field right
BYTE field number
TopView;field table
FCh set field table header
BYTE number of fields (must be <= existing number of fields)
BYTE screen behavior bits
bit 7 reserved
bit 6 set if menu items may be selected via keyboard
bit 5 set if left mouse button in "status" mode (press anywhere in window immediately returns control to app)
bit 4 set if right mouse button in "status" mode
bit 3 if set, select fields return contents or blanks rather than 'Y' or 'N'
bit 2 if set, modified bits reset on return to application
bits 0,1 = 00 no data returned on read of keyboard
01 data returned as array of characters containing all fields packed together, with no field numbers
10 data returned as numbered variable-length records for all fields
11 data returned as numbered variable-length records for the fields which were modified
BYTE current input field (updated by DESQview)
BYTE current select field (updated by DESQview)
BYTE attribute for select fields when they are pointed at
BYTE attribute for select fields which have been selected
FDh reset modified bit for all fields
FEh reset selected and modified bits for all fields
FFh set up input fields

## INT 15h Function 12h, Subfunction 05h

6 BYTEs table header (see opcode FCh above)
the field table entries, one for each field
BYTE start row
BYTE start column if menu selection and start is to right
BYTE end row or below end, selectable from keyboard only
BYTE end column
BYTE field type (see Table 44-18)
BYTE modifier (see Table 44-19 for fill-in fields)
if select field, first key to press to activate
00h if have to point-&-click or key is an extended-ASCII keystroke
(only if two-key menus enabled)
BYTE (select field only) normal color of field
BYTE second key for select field (this byte is present if and only if two-letter menu entries selected with opcode E5h, and in that case is present regardless of field type)

**Details:** DESQview uses and updates the actual copy of the information which is contained in the stream. Thus this info must remain intact until after the data entry is complete.

### 44-14 MODE 01h "Query Stream" Opcodes:args

A0h return logical cursor row in next byte
A1h return logical cursor column in next byte
A2h return top row of scrolling region in next byte
A3h return left column of scrolling region in next byte
A4h return row of physical window origin in next byte
A5h return column of physical window origin in next byte
A6h return height of physcial window in next byte
A7h return width of physical window in next byte
A8h return row of viewport origin in next byte
A9h return column of viewport origin in next byte
AAh return height of virtual screen in next byte
ABh return width of virtual screen in next byte
AFh return current video mode in next byte
C0h return current logical cursor position in next two bytes
C1h return top left corner of scrolling region in next two bytes
C2h return current window position in next two bytes
C3h return current window size in next two bytes
C4h return current viewport origin in next two bytes
C5h return current virtual screen size in next two bytes
D0h \ overwritten with D0h if frames may fall off screen edge
D1h / D1h if frames always displayed entirely
D2h \ overwritten with D2h if DESQview controls color palette
D3h / D3h if application changes color palette
D4h \ overwritten with D4h if window visible
D5h / D5h if window hidden
D6h \ overwritten with D6h if window has frame
D7h / D7h if window unframed
D8h \ overwritten with D8h if reading characters from window
D9h / D9h if reading attributes from window
DAh \ overwritten with DAh if using logical attributes
DBh / DBh if using physical attributes
DCh \ overwritten with DCh if TTY control char interpretation on

DDh / DDh if TTY control char interpretation off
DEh \ overwritten with DEh if writing both characters and attributes
DFh / DFh if leaving attributes untouched
E2h return current color in next byte
ECh get logical attributes for window contents
    BYTE execute call if currently in specified video mode
        bit 7 monochrome
        bit 6 color text, EGA/VGA graphics
        bit 5 medium-resolution CGA graphics
        bit 4 high-resolution CGA graphics
    BYTE which attributes to get
        bit 7 *unused*
        bits 4-6 first attribute to get - 1
        bits 0-3 # consecutive attributes
    N BYTEs buffer to hold attributes
EDh get logical attributes for window frame
    BYTE execute call if currently in video mode (see opcode ECh)
    BYTE which attributes to get
        bit 7 *unused*
        bits 4-6 first attribute to get - 1
        bits 0-3 # consecutive attributes
    N BYTEs buffer to hold attributes
EEh get characters for window frame
    BYTE execute call if currently in video mode (see opcode ECh)
    BYTE which attributes to get
        bit 7 *unused*
        bits 4-6 first char to get - 1
        bits 0-3 # consecutive characters
    N BYTEs buffer to hold characters
EFh return first N characters of current window name
    BYTE max length of returned name
    N BYTEs buffer to hold window name
F3h return contents of specified field
    BYTE field number
    N BYTEs buffer to hold field contents (size exactly equal to field size)
F5h get field table entry
    BYTE field number
    7-8 BYTEs buffer to hold field table entry
    **Notes:** DV < 2.26 always returns 7 bytes
        DV 2.26+ w/ APILEVEL < 2.26 returns 8 bytes iff field table is using 8-byte entries and eighth byte after F5h is E7h (NOP); otherwise, 7 bytes are returned
        DV 2.26+ w/ APILEVEL > 2.26 returns 7 or 8 bytes depending on the field table entry size
F6h get type of a field
    BYTE field number
    BYTE type
FCh get field table header

## INT 15h Function 12h, Subfunction 05h

---

**44-15 MODE 10h "Manager Stream" Opcodes:args**

00h allow window to be moved horizontally
01h allow window to be moved vertically
02h allow window to change width
03h allow window to change height
04h allow window to be scrolled horizontally
05h allow window to be scrolled vertically
06h allow "Close Window" menu selection for application
07h allow "Hide Window" menu selection for application
DESQview;suspending tasks
08h allow application to be suspended ("Rearrange/Freeze")
0Eh allow "Scissors" menu
10h allow DESQview main menu to be popped up
11h allow "Switch Windows" menu
12h allow "Open Window" menu
13h allow "Quit" menu selection
20h-33h opposite of 00h-13h, disallow specified action
40h notify if horizontal position of window changes
41h notify if vertical position of window changes
42h notify if width of window changes
43h notify if height of window changes
44h notify if window scrolled horizontally
45h notify if window scrolled vertically
46h notify if window is closed--program has to clean up and exit itself
47h notify if window is hidden
48h notify if "?" on main menu selected
49h notify if pointer message sent to window
4Ah notify if window is placed in foreground
4Bh notify if window is placed in background
4Ch notify if video mode changes
4Dh notify if "Scissors" menu "Cut" option selected
4Eh notify if "Scissors" menu "Copy" option selected
4Fh notify if "Scissors" menu "Paste" option selected
50h notify if DESQview main menu about to pop up
51h notify if DESQview main menu popped down
60h-71h opposite of 40h-51h: don't notify on specified event
84h attach window to parent task's window (both move together)
85h detach window from parent task's window (may move independently)
background processing;TopView
86h disable background operation for application
87h enable running in background
88h set minimum size of physical window
    BYTE rows
    BYTE columns
89h set maximum size of physical window
    BYTE rows
    BYTE cols
8Ah set primary asynchronous notification routine
    DWORD address of routine, 0000h:0000h means none (see also Table 44-20)

8Bh set asynchronous notification parameter
    DWORD 32-bit value passed to 8Ah async routine in DS:SI
DESQview;select field processing
ACh (DV2.2+) perform regular select field attribute processing
ADh (DV2.2+) protect attributes in selected field from being lost
DESQview;notification window
AEh make window default notify window for owning application (API level 2.00+)
AFh set selected field marker character
    BYTE character to display at left edge of selected fields
BCh set standard field processing mode
BDh set alternate field processing mode (enables cursor pad for menus)
BEh disables changing reverse logical attributes with ECh opcode
BFh enables changing reverse logical attributes with ECh opcode
C0h make current window topmost in system
C1h force current process into foreground
C2h make current window topmost in process
C3h position mouse pointer relative to origin of current field
    BYTE rows below upper left corner of field
    BYTE columns to right of upper left corner of field
C4h position mouse pointer relative to origin of given field
    BYTE field number
    BYTE rows below upper left corner of field
    BYTE columns to right of upper left corner of field
C5h orphan current window (also hides it)
**Note:** must be last in stream; all subsequent commands are ignored
C6h show all windows for this process
C7h hide all windows for this process
C8h suspend process and hide all its windows
C9h force current process into background
CAh make current window bottom-most in process
CBh cancel current window manager operation, remove DV menu, give control to topmost application
CCh orphan window and give it to the system for use as paste data
CEh reorder windows
    DWORD pointer to null-terminated list of words; each word is segment of object handle for a window
FFh no operation

### 44-16 MODES 14h to 1Fh "USER STREAMS":

**Registers at call:**
CX = number of bytes in stream
DS:SI -> first byte of actual stream (not header)
ES:DI = window's handle

**Return Registers:**
all registers unchanged

**Details:** These stream types are normally NOPs, but may be defined by the SETESC message to invoke FAR routines, one for each mode number, with registers set as shown above.

### 44-17 Logical attributes:

| | |
|---|---|
| 1 | normal text |
| 2 | highlighted normal text |
| 3 | help text |
| 4 | highlighted help text |

## INT 15h Function 12h, Subfunction 05h

| | |
|---|---|
| 5 | error message |
| 6 | highlighted error message |
| 7 | emphasized text |
| 8 | marked text |
| 9-16 | are reverse video versions of 1-8 |

### 44-18 Bitfields for input field types:

| | |
|---|---|
| bits 7,6 | type<br>00 inactive (non-entry) field<br>01 echos keystrokes input to make menu selection<br>10 fill-in field<br>11 select field |
| bit 5 | field can be filled by broadcast write (F7h opcode) |
| bit 4 | reserved |
| bit 3 | reserved |
| bit 2 | reserved |
| bit 1 | set if field selected |
| bit 0 | set if field modified |

### 44-19 Bitfields for fill-in field modifiers:

| | |
|---|---|
| bit 7 | if set, automatically enter CR when field full |
| bit 6 | move to next field when current field is full |
| bit 5 | if set, enter text from right end (for numbers) |
| bit 4 | if set, force input to uppercase |
| bit 3 | if set, clear old contents on first keystroke |
| bit 2 | if set, input returned when cursor moves out of modified field ("validate",API level 2.02+) |
| bit 1 | reserved |
| bit 0 | reserved |

### 44-20 Asynchronous notification routine defined by manager stream 8Ah calling convention:

| Registers at call: | Return Registers: |
|---|---|
| ES:DI = handle of window | all registers unchanged |
| DS:SI is 32-bit value set by 8Bh manager stream opcode | |

### 44-21 Mailbox message indicating event:

Opcode

40h horizontal movement

| | |
|---|---|
| DWORD | object handle of window |
| BYTE | new row |
| BYTE | new col |

41h vertical movement

| | |
|---|---|
| DWORD | object handle of window |
| BYTE | new row |
| BYTE | new col |

42h horizontal size change

| | |
|---|---|
| DWORD | object handle of window |
| BYTE | new rows |
| BYTE | new cols |

43h vertical size change

| | |
|---|---|
| DWORD | object handle of window |
| BYTE | new rows |
| BYTE | new cols |

44h scrolled horizontally

| | |
|---|---|
| DWORD | object handle of window |
| BYTE | mouse row within window |
| BYTE | mouse column within window |
| BYTE | field mouse is on, 0 if none |
| BYTE | amount moved: >0 right, <0 left, 0 done |

45h scrolled vertically

| | |
|---|---|
| DWORD | object handle of window |
| BYTE | mouse row within window |
| BYTE | mouse column within window |
| BYTE | field mouse is on, 0 if none |
| BYTE | amount moved: >0 down, <0 up, 0 done |

46h window close request

| | |
|---|---|
| DWORD | object handle of window |
| BYTE | mouse pointer row |
| BYTE | mouse pointer column |
| BYTE | field mouse is on, 0 if none |

47h application's windows hidden

48h Help for Program selected

| | |
|---|---|
| DWORD | object handle of window |
| BYTE | mouse pointer row |
| BYTE | mouse pointer column |
| BYTE | field mouse is on, 0 if none |

49h pointer message sent to window

| | |
|---|---|
| DWORD | pointer handle which received message |

4Ah switched to window from another ("raise")

4Bh switched away from the window ("lower")

4Ch video mode changed

| | |
|---|---|
| BYTE | new BIOS video mode |

4Dh Scissors/cUt selected

| | |
|---|---|
| DWORD | object handle of window |
| BYTE | row of upper left corner |
| BYTE | column of upper left corner |
| BYTE | field number ul corner is in, 0=none |
| DWORD | handle of orphaned window created with copy of data from specified region |
| BYTE | height of region |
| BYTE | width of region |

4Eh Scissors/Copy selected

| | |
|---|---|
| DWORD | object handle of window |
| BYTE | row of upper left corner |
| BYTE | column of upper left corner |
| BYTE | field number ul corner is in, 0=none |
| DWORD | handle of orphaned window created with copy of data from specified region |
| BYTE | height of region |
| BYTE | width of region |

4Fh Scissors/Paste selected

| | |
|---|---|
| DWORD | object handle of window |
| BYTE | row of upper left corner |
| BYTE | column of upper left corner |

## INT 15h Function 12h, Subfunction 05h

| | |
|---|---|
| BYTE | field number ul corner is in, 0=none |
| DWORD | handle of orphaned window with data |
| BYTE | height of region |
| BYTE | width of region |

**Note:** orphaned data window should be adopted or freed when done

50h main menu about to pop up

51h main menu popped down

---

## INT 15h Function 12h, Subfunction 08h

*TopView*

### *SEND MESSAGE - "SIZEOF" - GET OBJECT SIZE*

**Registers at call:**

AH = 12h

BH = 08h

BL = handle identifier (see Table 44-5)

- window: total character positions in window
- timer: elapsed time since timer started
- pointer: number of messages queued to pointer object
- panel: number of panels in panel file
- keyboard: number of input buffers queued
- mailbox: number of messages queued
- objectq: number of objects queued

STACK: DWORD object's handle (if BL even)

**Return Registers:**

STACK: DWORD result

**Details:** For panel objects, a count of zero is returned if no panel file is open for the object.

**See also:** Function 12h Subfunctions 04h and 09h

---

## INT 15h Function 12h, Subfunction 09h

*TopView*

### *SEND MESSAGE - "LEN" - GET OBJECT LENGTH*

**Registers at call:**

AH = 12h

BH = 09h

BL = object

- 00h handle in DWORD on top of stack
  - window: get chars/line
  - timer: get 1/100 sec until timer expires
  - mailbox: (DV/X) number of bytes queued
- 01h current task's default window
- 0Ch (DV 2.26+) window owning handle on stack
- 0Dh (DV 2.26+) parent task's window

**Return Registers:**

STACK: DWORD length

**See also:** Function 12h Subfunction 08h

---

## INT 15h Function 12h, Subfunction 0Ah

*TopView*

### *SEND MESSAGE - "ADDTO" - WRITE CHARACTERS AND ATTRIBUTES TO WINDOW*

**Registers at call:**

AH = 12h

BH = 0Ah

BL = window to which to write

- 00h window handle is DWORD on top of stack
- 01h current task's default window
- 0Ch (DV 2.26+) default window of task owning handle on top of stack
- 0Dh (DV 2.26+) default window of current task's parent

STACK: DWORD count of attributes
DWORD address of attribute string

**Return Registers:**
DWORD count of characters
STACK popped
DWORD address of character string

**Details:** If one string is longer than the other, the shorter one will be reused until the longer one is exhausted.

The cursor is left just after the last character written.

**See also:** Function 12h Subfunction 0Bh (WINDOW)

---

### INT 15h Function 12h, Subfunction 0Ah

*TopView*

### *SEND MESSAGE - "ADDTO" - SEND MAILBOX MESSAGE/STATUS BY VALUE*

**Registers at call:**
AH = 12h
BH = 0Ah
BL = mailbox to write to
  00h handle is DWORD on top of stack
  02h default mailbox of task whose handle is on top of stack
  03h current task's default mailbox
STACK: DWORD status (low byte)
  DWORD length of message
  DWORD address of message

**Return Registers:**
STACK popped

**Details:** The message is copied into either system or common memory. Insufficient memory normally causes the process to be aborted; under DESQview 2.2+, failed writes may return CF set instead (see Function DE15h).

**See also:** Function 12h Subfunction 0Bh (MAILBOX)

---

### INT 15h Function 12h, Subfunction 0Ah

*TopView*

### *SEND MESSAGE - "ADDTO" - SET OBJECT BITS*

**Registers at call:**
AH = 12h
BH = 0Ah
BL = handle identifier (see Table 44-5)
  timer: start timer for specified interval
  pointer: set control flags
  keyboard: set control flags
STACK: DWORD object's handle (if BL even)
  (timer) DWORD duration in 1/100 seconds
  (else) DWORD bits to set

**Return Registers:**
STACK popped

**See also:** Function 12h Subfunction 0Bh (OBJECT)

**44-22 Bitfields for keyboard objects:**

| | |
|---|---|
| bit 15 | reserved, can't be set |
| bit 14 | unused |
| bit 13 | reserved, can't be set |
| bit 12-6 | unused |
| bit 5 | (DV 2.2+) exclusive input |
| bit 4 | filter all keys (used with handler established by SETESC)<br>if 0, only keys that would normally be displayed are filtered |
| bit 3 | program continues executing while input in progress |
| bit 2 | insert mode active for field mode |
| bit 1 | hardware cursor displayed when task is hardware cursor owner<br>must be set if keyboard in field mode and field table includes input fields |
| bit 0 | keyboard is in field mode rather than keystroke mode |

**44-23 Bitfields for pointer objects:**

| | |
|---|---|
| bit 15 | reserved, can't be set |
| bit 14-8 | unused |
| bit 7 | mouse pointer is hidden while in window |
| bit 6 | get messages even if window not topmost |
| bit 5 | get messages even if window not foreground |
| bit 4 | multiple clicks separated by less than 1/3 second are counted and returned in a single message |
| bit 3 | pointer position is relative to screen origin, not window origin |
| bit 2 | send message on button release as well as button press |
| bit 1 | (DV 2.23+) send message with row=FFFFh and col=FFFFh whenever pointer leaves the window |
| bit 0 | send message only on button activity, not movement<br>(DV-specific, and INT 15h Function DE0Fh must have been called first) |

---

## INT 15h Function 12h, Subfunction 0Bh *TopView*

### *SEND MESSAGE - "SUBFROM" - WRITE ATTRIBUTES TO WINDOW*

**Registers at call:**
AH = 12h
BH = 0Bh
BL = handle identifier of window (see Table 44-5)
STACK: DWORD window's handle (if BL even)
DWORD number of attributes to write
DWORD address of attributes

**Return Registers:**
STACK popped

**Details:** The attributes are written starting at the current cursor position; the cursor is left just after the last position written.

**See also:** Function 12h Subfunction 0Ah (WINDOW)

---

## INT 15h Function 12h, Subfunction 0Bh *TopView*

### *SEND MESSAGE - "SUBFROM" - SEND MAILBOX MESSAGE/STATUS BY REFERENCE*

**Registers at call:**
AH = 12h
BH = 0Bh
BL = mailbox to which to write
  00h handle is DWORD on top of stack
  02h default mailbox of task whose handle is on top of stack
  03h current task's default mailbox
STACK: DWORD mailbox handle (if BL even)
DWORD status (low byte)
DWORD length of message
DWORD address of message

**Return Registers:**
STACK popped

**Details:** Only a pointer to the message is stored, but the write may still fail due to insufficient memory. Under DV 2.2+, failed mailbox writes may return CF set (see Function DE15h).

**See also:** Function 12h Subfunction 0Ah (MAILBOX)

---

## INT 15h Function 12h, Subfunction 0Bh *TopView*

### *SEND MESSAGE - "SUBFROM" - REMOVE OBJECT FROM OBJECTQ*

**Registers at call:**
AH = 12h
BH = 0Bh
BL = OBJECTQ from which to remove all copies of a particular object
  06h OBJECTQ of task whose handle is on top of stack
  07h task's default OBJECTQ

**Return Registers:**
STACK popped

STACK: DWORD task handle (if BL=06h)
DWORD handle of object to remove

**Details:** This message should be sent whenever an object is erased or closed.

---

### INT 15h Function 12h, Subfunction 0Bh
### *SEND MESSAGE - "SUBFROM" - RESET OBJECT BITS*

*TopView*

**Registers at call:**
AH = 12h
BH = 0Bh
BL = handle identifier (see Table 44-5)
pointer: reset control flags
keyboard: reset control flags
STACK: DWORD object's handle (if BL even)
DWORD which bits to clear (see Tables 44-24 and 44-25)

**Return Registers:**
STACK popped

**See also:** Function 12h Subfunction 0Ah (OBJECT)

---

### INT 15h Function 12h, Subfunction 0Ch
### *SEND MESSAGE - "OPEN" - OPEN OBJECT*

*TopView*

**Registers at call:**
AH = 12h
BH = 0Ch
BL = handle identifier (see Table 44-5)
window: fill with given character from scroll origin to end
keyboard: attach to a window
timer: open
pointer: start taking input for window
panel: associate with a panel file
STACK: DWORD object's handle (if BL even)
(windw) DWORD character to fill with
(kybd) DWORD handle of window to attach to
(ptr) DWORD handle of window to attach to
(panel) DWORD length of filename or resident panel
DWORD address of filename or resident panel
(else) nothing

**Return Registers:**
STACK popped

**Details:** If the first byte of the panel file name is 1Bh, then the "name" *is* a panel. If the first two bytes of the panel file name are C0hC3h, then the "name" *is* the panel file.

The result code of the open may be retrieved with the STATUS message.

The logical cursor is left at the scroll origin after filling a window.

The task opening a mailbox becomes its owner, and the only task allowed to read the mailbox.

Messages are only sent to a pointer object when the mouse is positioned in the window to which the pointer has been attached.

There is no need to explicitly open a timer object, as ADDTO and WRITE messages automatically open the timer.

**See also:** Function 12h Subfunctions 0Dh and 14h (LOCK)

---

### INT 15h Function 12h, Subfunction 0Dh
### *SEND MESSAGE - "CLOSE" - CLOSE OBJECT*

*TopView*

**Registers at call:**
AH = 12h
BH = 0Dh

**Return Registers:**
STACK popped if parameter passed on stack

BL = handle identifier (see Table 44-5)
  timer or panel: close
  keyboard: detach from window and discard queued input
  pointer: stop taking input
  mailbox: close, unlock, and discard any pending messages
STACK: DWORD object's handle (if BL even)

**Details:** When an OBJECTQ is closed, each object in the OBJECTQ is sent an ERASE message (Function 12h Subfunction 0Eh).

When a panel object is closed, the panel file and any panels currently in use are freed; window and keyboard objects created by APPLY are not affected, but field mode input ceases.

Open but idle timer objects consume a small amount of CPU time.

**See also:** Function 12h Subfunctions 0Ch, 0Eh, and 14h (LOCK)

---

## INT 15h Function 12h, Subfunction 0Eh

*TopView*

### *SEND MESSAGE - "ERASE" - ERASE OBJECT*

**Registers at call:**
AH = 12h
BH = 0Eh
BL = handle identifier (see Table 44-5)
  window: clear from scroll origin to end of window
  keyboard: discard input
  timer: cancel current interval
  pointer: discard all pending messages
  mailbox: discard all pending messages
  objectq: remove all objects from queue
STACK: DWORD object's handle (if BL even)

**Return Registers:**
STACK popped if parameter passed on stack

**Details:** When an OBJECTQ is erased, each object in the OBJECTQ is also erased.

**See also:** Function 12h Subfunction 02h

---

## INT 15h Function 12h, Subfunction 0Fh

*TopView*

### *SEND MESSAGE - "STATUS" - GET OBJECT STATUS*

**Registers at call:**
AH = 12h
BH = 0Fh
BL = handle identifier (see Table 44-5)
  timer: is it running?
  pointer: return status of last message
  panel: verify success of last OPEN or APPLY
  mailbox: status of last message read
  keyboard: status of last message read
  ObjectQ: determine whether it is open
STACK: DWORD object's handle (if BL even)

**Return Registers:**
STACK: DWORD status

**Details:** If the object is a panel object, the status indicates the error code (see Table 44-24).

If the object is a timer, the status is:
  00000000h open but not running
  40000000h open and running
  80000000h closed.

If the object is an OBJECTQ, the status is: 00000000h open, 80000000h closed.

If the object is a keyboard in keystroke mode, the status is the extended character code (scan code) of the last keystroke.

If the object is a keyboard in field mode, the status indicates the reason for the last return from the field manager (see Table 44-25).

The status of mailbox messages sent by the window manager is always 80h.

The status of a pointer message is the same as the status field in the message.

**See also:** Function 12h Subfunction 04h (READ)

### 44-24 Panel error codes:

| | | |
|---|---|---|
| 00h | successful | |
| 14h | panel name not in panel directory | |
| 15h | not enough memory to apply panel | |
| 16h | invalid panel format | |
| 17h | panel file already open | |
| 81h-92h | DOS error codes+80h | \ codes > 80h indicate |
| 95h | not enough memory to open panel file | > that the panel was |
| 98h | null panel file name | / not opened |

### 44-25 Reasons for returning from field manager:

| | |
|---|---|
| 00h | Enter key pressed |
| 01h | Button 1 or keystroke selection |
| 02h | Button 2 |
| 03h | validation |
| 04h | automatic Enter on field |
| 1Bh | Escape pressed |
| 46h | ^Break pressed |
| other | extended code for key terminating input. |

## INT 15h Function 12h, Subfunction 10h

*TopView*

### *SEND MESSAGE - "EOF" - GET OBJECT EOF STATUS*

**Registers at call:**
AH = 12h
BH = 10h
BL = handle identifier (see Table 44-5)
window: return TRUE if logical cursor past end of window
mailbox: *unknown*
STACK: DWORD object's handle (if BL even)

**Return Registers:**
STACK: DWORD status

## INT 15h Function 12h, Subfunction 11h

*TopView*

### *SEND MESSAGE - "AT" - POSITION OBJECT CURSOR*

**Registers at call:**
AH = 12h
BH = 11h
BL = handle identifier for window (see Table 44-5)
STACK: DWORD object's handle (if BL even)
DWORD column
DWORD row

**Return Registers:**
STACK popped

## INT 15h Function 12h, Subfunction 11h

*TopView*

### *SEND MESSAGE - "SETNAME" - ASSIGN NAME TO MAILBOX*

**Registers at call:**
AH = 12h
BH = 11h
BL = handle identifier for mailbox (see Table 44-5)
STACK: DWORD object's handle (if BL even)

DWORD length of name
DWORD address of name

**Return Registers:**
STACK popped

**See also:** DESQview Function 12h Subfunction 12h (GETNAME), DESQview Function DE0Eh (below)

---

### INT 15h Function 12h, Subfunction 1100h

*TopView*

### *SEND MESSAGE - "SETSCALE" - SET POINTER SCALE FACTOR*

**Registers at call:**
AH = 12h
BX = 1100h
STACK: DWORD object handle for pointer object
DWORD number of colums to scale pointer position to
DWORD number of rows to scale pointer position to

**Return Registers:**
STACK popped

**See also:** Function 12h Subfunction 1200h

---

### INT 15h Function 12h, Subfunction 12h

*TopView*

### *SEND MESSAGE - "READN" - GET NEXT N OBJECT BYTES*

**Registers at call:**
AH = 12h
BH = 12h
BL = handle identifier for window (see Table 44-5)
STACK: DWORD object's handle (if BL even)
DWORD count to read

**Return Registers:**
STACK: DWORD number of bytes actually read
DWORD address of buffer containing data

**Details:** Reading starts at the current logical cursor position; the cursor is updated to point at the character following the last one read.

Any translucent blanks (FFh) which are visible on screen are changed to the character which is seen through them.

The string produced by the read is placed in an input buffer which may be reused by the next READ or READN of a window.

Window stream opcodes D8h and D9h determine whether the read returns characters or attributes.

**See also:** Function 12h Subfunctions 04h (WINDOW) and 05h (WINDOW)

---

### INT 15h Function 12h, Subfunction 1200h

*TopView*

### *SEND MESSAGE - "GETSCALE" - GET POINTER SCALE FACTOR*

**Purpose:** Determine the virtual window size currently used for purposes of scaling the pointer's position. A pointer which is currently in the lower right-hand corner of the window will report its position as the virtual window size rather than the actual window size on screen.

**Registers at call:**
AH = 12h
BX = 1200h
STACK: DWORD object handle for pointer

**Return Registers:**
STACK: DWORD columns
DWORD rows

**See also:** Function 12h Subfunction 1100h

---

### INT 15h Function 12h, Subfunction 13h

*TopView*

### *SEND MESSAGE - "REDRAW" - REDRAW WINDOW*

**Registers at call:**
AH = 12h
BH = 13h
BL = handle identifier for window (see Table 44-5)
STACK: DWORD object's handle (if BL even)

**Return Registers:**
STACK popped if parameter passed on stack

**See also:** Function 12h Subfunctions 05h (WINDOW) and 0Eh

---

### INT 15h Function 12h, Subfunction 1300h

*TopView*

***SEND MESSAGE - "SETICON" - SPECIFY POINTER ICON***

**Registers at call:**
AH = 12h
BX = 1300h
STACK: DWORD object handle for pointer
DWORD character to use for pointer

**Return Registers:**
STACK popped

**Details:** DESQview accepts but ignores this call.

---

### INT 15h Function 12h, Subfunction 14h

*TopView*

***SEND MESSAGE - "SETESC" - SET ESCAPE ROUTINE ADDRESS***

**Registers at call:**
AH = 12h
BH = 14h
BL = message modifier
    00h handle is DWORD on top of stack
    01h define user stream
    04h intercept keystrokes from KEYBOARD to window (handle on stack)
    05h intercept keystrokes from task's default KEYBOARD to a window
STACK: DWORD object's handle (if BL even)
(windw) DWORD user stream number (14h-1Fh)
DWORD address of FAR user stream handler
(keybd) DWORD address of FAR filter function

**Return Registers:**
STACK popped

#### 44-26 Keyboard filter function calling convention:

**Registers at call:**
AL = character
AH = 00h or extended ASCII code if AL = 00h
BL = field number
CH = cursor column
CL = cursor row
DL = field type modifier (sixth item in field table entry)
DH = seventh item in field table entry
ES:SI = window's handle
DS:DI -> field table entry for field containing the cursor

**Return Registers:**
AH = action
    00h use keystroke
    01h ignore keystroke
    FFh beep and ignore keystroke

**Details:** The keyboard filter function is called when the keyboard is in field mode. It is not allowed to make INT 15h, DOS, or BIOS calls.

---

### INT 15h Function 12h, Subfunction 14h

*TopView*

***SEND MESSAGE - "LOCK" - REQUEST EXCLUSIVE ACCESS TO RESOURCE***

**Registers at call:**
AH = 12h
BH = 14h
BL = object
    00h mailbox handle is DWORD on top of stack
    02h use given task's mailbox (task's handle on top of stack)
    03h use current task's default mailbox
STACK: DWORD object's handle (if BL even)

**Return Registers:**
STACK popped if parameter passed on stack

**Details:** Release the exclusive access by sending a CLOSE message to the mailbox.

Access may be requested multiple times, and requires multiple CLOSEs.

**See also:** Function 12h Subfunction 0Dh

## DESQview

While it is almost impossible to find a TopView application in use today, Quarterdeck's DESQview is very much alive and well (even without the marketing clout of a Microsoft or an IBM behind it). This chapter describes the additional functionality DESQview provides over the TopView API; except as noted, these functions are also supported by DESQview/X.

All of the INT 15h Function 10XXh, 11XXh, and 12h functions in this section conflict with MultiDOS Plus (Chapter 45) and VMiX (Chapter 46).

---

### INT 10h Function 80h
### *SET UNKNOWN HANDLER*

*DESQview 2.0x only (undocumented)*

**Registers at call:**
AH = 80h
DX = 4456h ('DV')
ES:DI -> FAR subroutine to be called on *unknown*

**Return Registers:**
DS = segment of DESQview data structure for video buffer

**Details:** This function is probably meant for internal use only, due to the magic value required in DX.

The subroutine seems to be called when the DESQview menu is accessed; on entry, AL = 03h or 04h.

**Conflicts:** CU Writer (Chapter 66)

---

### INT 10h Function 81h
### *GET UNKNOWN ADDRESS*

*DESQview 2.0x only (undocumented)*

**Registers at call:**
AH = 81h
DX = 4456h ('DV')

**Return Registers:**
ES = segment of DESQview data structure for video buffer
  BYTE ES:[0] = current window number in DV 2.0x

**Details:** This function is probably meant for internal use only, due to the magic value required in DX.
**Conflicts:** CU Writer (Chapter 66)
**See also:** Function 82h

---

### INT 10h Function 82h
### *GET CURRENT WINDOW INFO*

*DESQview 2.0x only (undocumented)*

**Registers at call:**
AH = 82h
DX = 4456h ('DV')

**Return Registers:**
DS = segment in DESQview for data structure
  in DV 2.00,
    BYTE DS:[0] = window number
    WORD DS:[1] = segment of other data structure
    WORD DS:[3] = segment of window's object handle
ES = segment of DESQview data structure for video buffer
AL = current window number
AH = *unknown*
BL = direct screen writes
  00h program does not do direct writes
  01h program does direct writes, so shadow buffer not usable
BH = *unknown*
CL = current video mode
CH = *unknown*

**Details:** This function is probably meant for internal use only, due to the magic value required in DX.
**Conflicts:** AX PC (Chapter 6)
**See also:** Function 81h

## INT 15h Function 102Ch

*DESQview v2.00+*

### *"NEWPROC" - START NEW APPLICATION IN NEW PROCESS*

**Registers at call:**
AX = 102Ch
ES:DI -> contents of .PIF/.DVP file (see Table 44-25)
BX = size of .PIF/.DVP info

**Return Registers:**
BX = segment of object handle for new task
(0000h on error)

**See also:** Function DE24h, MS-DOS INT 21h Function 4Bh (Chapter 24)

**44-27 Format of .PIF/.DVP file:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | reserved (0) |
| 01h | BYTE | checksum of bytes 02h through 170h |
| 02h | 30 BYTEs | blank-padded program title |
| 20h | WORD | maximum memory to allocate to partition in KB |
| 22h | WORD | minimum memory required in KB |
| 24h | 64 BYTEs | ASCIZ program pathname |
| 64h | BYTE | default drive letter ('A',...) |
| 65h | 64 BYTEs | ASCIZ default directory name |
| A5h | 64 BYTEs | ASCIZ program parameters |
| E5h | BYTE | initial screen mode (0-7) (see also offset 189h) |
| E6h | BYTE | number of text pages used |
| E7h | BYTE | number of first interrupt to save |
| E8h | BYTE | number of last interrupt to save |
| E9h | BYTE | rows in virtual screen buffer |
| EAh | BYTE | columns in virtual screen buffer |
| EBh | BYTE | initial window position, row |
| ECh | BYTE | initial window position, column |
| EDh | WORD | system memory in KB |
| EFh | 64 BYTEs | ASCIZ shared program name |
| 12Fh | 64 BYTEs | ASCIZ shared program data file |
| 16Fh | BYTE | flags1<br>bit 7: writes text directly to screen<br>bit 6: runs in foreground only (see also offset 184h)<br>bit 5: uses math coprocessor<br>bit 4: accesses system keyboard buffer directly<br>bits 3-1: reserved (0)<br>bit 0: swappable |
| 170h | BYTE | flags2<br>bit 6: uses command-line parameters in field at A5h<br>bit 5: swaps interrupt vectors |
| **---information unique to .DVP files---** | | |
| 171h | 2 BYTEs | keys to use on open menu |
| 173h | WORD | size of script buffer in bytes |
| 175h | WORD | automatically give up CPU after this many tests for keyboard input in one clock tick (default 0 = never) |
| 177h | BYTE | nonzero = "uses own colors" |
| 178h | BYTE | nonzero if application swappable |
| 179h | 3 BYTEs | reserved (0) according to Quarterdeck documentation<br>in actual .DVP files, frequently 01h |
| 17Ch | BYTE | nonzero to automatically close on exit (see also offset 18Bh) |

| | | |
|---|---|---|
| 17Dh | BYTE | nonzero if copy-protect floppy is required |
| **---information unique to DESQview 2.0+---** | | |
| 17Eh | BYTE | .DVP version number |
| | | 00h DESQview v1.2+ |
| | | 01h DESQview v2.0+ |
| | | 02h DESQview v2.2+ |
| 17Fh | BYTE | reserved (0) |
| 180h | BYTE | initial number of rows in physical window |
| 181h | BYTE | initial number of columns in physical window |
| 182h | WORD | maximum expanded memory to allow, in KB |
| 184h | BYTE | flags3 |
| | | bit 7: automatically assign window position |
| | | bit 5: maximum memory value has been specified |
| | | bit 4: disallow "Close" command |
| | | bit 3: foreground-only when doing graphics |
| | | set by DV 2.3+ when "Runs in Background" = "D" (undoc) |
| | | bit 2: don't virtualize (see also offset 18Bh) |
| | | bit 1: foreground-only during DOS calls |
| | | set by DV 2.3+ when "Runs in Background" = "D" (undoc) |
| 185h | BYTE | keyboard conflict level (0-4 for DV<2.26, 00h-0Fh for DV2.26+) |
| | | bit 3: save/restore entire INT 09h handler state every taskswtch |
| | | bit 2: take special precautions for programs which read the BIOS keyboard buffer directly from memory |
| | | bit 1: never indicate keystroke available during scripts/xfers |
| | | bit 0: only indicate keystroke available every sixth poll |
| 186h | BYTE | number of graphics pages used |
| 187h | WORD | extra system memory size |
| 189h | BYTE | initial screen mode (FFh = default) (overrides offset E5h) |
| **---information unique to DESQview 2.2+---** | | |
| 18Ah | BYTE | serial port usage |
| | | FFh uses all serial ports |
| | | 00h no serial ports |
| | | 01h only COM1 |
| | | 02h only COM2 |
| 18Bh | BYTE | flags4 |
| | | bit 7: automatically close application on exit if .COM or .EXE specified (see also offset 17Ch) |
| | | bit 6: swappable if not using serial ports |
| | | bit 5: start program with window hidden (v2.26+) |
| | | bit 4: start program in background (v2.26+) |
| | | bit 3: virtualize text (see also offset 184h) |
| | | bit 2: virtualize graphics (see also offset 184h) |
| | | bit 1: share CPU when foreground |
| | | bit 0: share EGA when foreground and zoomed |
| 18Ch | BYTE | protection level for 386 machines |
| 18Dh | 19 BYTEs | reserved (0) for regular DESQview |
| **---information unique to DESQview/X 1.0---** | | |
| 18Dh | BYTE | X flags |
| | | bits 3-7: unused (0) |

| | | |
|---|---|---|
| | | bit 2: don't display wait message when opening window<br>bit 1: don't display DOS window<br>bit 0: (XNEWPROC) use DOS client layer (DOS-to-X)<br>(NEWPROC) inherit DOS client layer usage |
| 18Eh | BYTE | X keyboard behavior (0-3) |
| 18Fh | BYTE | font scaling<br>00h fixed fonts<br>01h scalable fonts |
| 190h | 10 BYTEs | reserved (0) |
| 19Ah | WORD | length of data follownig XDVP signature |
| 19Ch | 4 BYTEs | signature "XDVP" |
| 1A0h | N BYTEs | list of variable length records (see Table 44-28) |

**44-28 Format of variable length record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | length of following record, 0000h if end of record list |
| 02h | BYTE | record type<br>01h script filename, up to 64 characters<br>02h command-line parameters (allows >64 characters on cmdline)<br>03h environment inheritance<br>04h environment string<br>05h starting window position |
| ---types 01h,02h,04h--- | | |
| 03h | N BYTEs | ASCII data |
| ---type 03h--- | | |
| 03h | BYTE | inheritance<br>00h do not inherit<br>01h inherit environment |
| ---type 05h--- | | |
| 03h | N BYTEs | ASCII copy of fields as typed into DVPMAN, separated by commas:<br>starting row, starting column, starting height, starting width |

**Note:** if there are multiple occurrences of record types 01h, 02h, or 03h, only the last instance of each type is used; multiple occurrences of type 04h are concatenated.

## INT 15h Function 102Dh
## *"KMOUSE" - KEYBOARD MOUSE CONTROL*

*DESQview v2.00+*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 102Dh | |
| BL = subfunction | |
| 00h determine whether using keyboard mouse | BL = result<br>00h using real mouse<br>01h using keyboard mouse |
| 01h turn keyboard mouse on | |
| 02h turn keyboard mouse off | |

## INT 15h Function 102Eh
## *ALLOCATE SYSTEM MEMORY*

*DESQview v2.40+*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 102Eh | AX = status |
| BX = number of bytes | 0000h successful<br>ES:DI -> allocated system memory (see Table 44-27) |

0001h failed
ES:DI = 0000h:0000h

**Details:** Under DESQview v2.42, this call is identical to Function 1001h.

**See also:** Functions 1001h, 1002h, and DE0Ch

**44-29 Format of system memory block header (one paragraph before memory block):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of next header or 0000h |
| 02h | WORD | segment of previous header or 0000h |
| 04h | WORD | size of block in paragraphs, including header |
| 06h | BYTE | availability flag (00h in use, 01h free) |

---

### INT 15h Function 1117h, Subfunction 0000h *DESQview v2.20+*
### *"ASSERTMAP" - GET/SET MAPPING CONTEXT*

**Registers at call:**
AX = 1117h
BX = mapping context
  0000h get current mapping context without setting
  nonzero set new mapping context

**Return Registers:**
BX = mapping context in effect before call
interrupts enabled

**Details:** This function differs from Function DE17h for DESQview v2.20 through 2.25.

Mapping contexts determine conventional-memory addressability; setting a mapping context ensures that the associated program and data areas are in memory for access. This function is usable by drivers, TSRs and shared programs; the caller need not be running under DESQview, but must ensure that the stack in use will not be mapped out by the call.

**See also:** Function DE17h, MS Windows INT 2Fh Function 1685h (Chapter 43)

---

### INT 15h Function 12h, Subfunction 03h *DESQview v2.26+*
### *"CONNECT" - CONNECT TWO WINDOWS*

**Registers at call:**
AH = 12h
BH = 03h
BL = window to be connected
  00h handle of window to be attached in DWORD on top of stack
  01h attach current task's main window
STACK: DWORD window's handle (only if BL=00h)
  DWORD handle of window to attach to or 00000000h to detach

**Return Registers:**
STACK popped

**Details:** When two windows are connected, both will move if the user moves either.

Multiple windows may be attached to a single window, but each window may only be attached to one window at a time.

**See also:** Function 12h Subfunction 83h

---

### INT 15h Function 12h, Subfunction 06h *DESQview 2.20+*
### *SEND MESSAGE - "SETPRI" - SET PRIORITY WITHIN OBJECTQ*

**Registers at call:**
AH = 12h
BH = 06h
BL = handle identifier (see Table 44-5)
  mailbox, keyboard, pointer, or timer
  (DV 2.50+) window
STACK: DWORD object's handle (if BL even)
  DWORD new priority of object in task's OBJECTQ (new priority of task if window handle)

**Return Registers:**
STACK popped

**Details:** Initially all objects have the same default value. Applications should only make relative adjustments to this default value.

When changing priorities, all objects already on the objectq are reordered.

For window handles, only the non-blocked task(s) with the highest priority receive CPU time under DESQview 2.50-2.52; the default priority is 0Ah.

**See also:** Function 12h Subfunction 07h and 87h

---

### INT 15h Function 12h, Subfunction 07h

*DESQview 2.20+*

***SEND MESSAGE - "GETPRI" - GET PRIORITY WITHIN OBJECTQ***

**Registers at call:**
AH = 12h
BH = 07h
BL = handle identifier (see Table 44-5)
  mailbox, keyboard, pointer, or timer
  (DV 2.50+) window
STACK: DWORD object's handle (if BL even)

**Return Registers:**
STACK: DWORD object priority

**Details:** Initially all objects have the same default value. Applications should only make relative adjustments to this default value.

**See also:** Function 12h Subfunction 06h

---

### INT 15h Function 12h, Subfunction 12h

*DESQview 2.50+*

***SEND MESSAGE - "GETNAME" - GET NAME OF MAILBOX***

**Registers at call:**
AH = 12h
BH = 12h
BL = handle identifier for mailbox (see Table 44-5)
STACK: DWORD mailbox handle (if BL even)
  DWORD length of buffer for name
  DWORD pointer to buffer

**Return Registers:**
STACK: DWORD length of returned name (or size of buffer, if less)

**Details:** The returned name is not NUL-terminated.

**See also:** Function 12h Subfunction 11h (SETNAME), Function DE0Eh

---

### INT 15h Function 12h, Subfunction 13h

*DESQview 2.50+*

***SEND MESSAGE - "READINTO" - GET NEXT MAIL MESSAGE***

**Registers at call:**
AH = 12h
BH = 13h
BL = handle identifier for mailbox (see Table 44-5)
STACK: DWORD mailbox handle (if BL even)
  DWORD size of buffer in bytes
  DWORD pointer to buffer

**Return Registers:**
STACK: DWORD number of bytes read

**Details:** This call blocks if no input is available, but will return less than the requested number of bytes if some (but insufficient) data is available.

Use this call instead of Function 12h Subfunction 04h if the mailbox has flag bits 4 or 5 set, as common memory may be exhausted by that call when attempting to read the next message.

**See also:** Function 12h Subfunctions 04h (READ), 05h, and 16h

---

### INT 15h Function 12h, Subfunction 15h

*DESQview v2.20+*

***SEND MESSAGE - "SETFLAGS" - SET OBJECT FLAGS***

**Registers at call:**
AH = 12h
BH = 15h
BL = handle identifier (see Table 44-5)

mailbox, keyboard, or pointer only
STACK: DWORD object's handle (if BL even)
DWORD flags (see Table 44-30)

**Return Registers:**
STACK popped

**Details:** This function is only available if the API level has been set to at least 2.20. It is equivalent to performing SUBFROM and ADDTO calls on the object.

If a mailbox has bits 4 or 5 set, you must use READINTO rather than READ (see Function 12h Subfunction 1300h) to retrieve messages.

**See also:** Function 12h Subfunctions 0Ah, 0Bh, and 16h

### 44-30 Bitfields for object flags:

| | |
|---|---|
| **mailbox** | |
| bit 0 | all mail messages in common memory |
| bit 1 | allow write even if closed |
| bit 2 | don't erase messages when mailbox closed |
| bit 4 | (DV/X) append messages with like status and sender (stream-oriented mail) |
| bit 5 | (DV/X) store mail in expanded memory (pool grows as needed) |
| bit 6 | (DV/X) make mailbox into non-owned mailbox |
| **keyboard** | |
| bit 5 | exclusive input when keyboard in use for input |

## INT 15h Function 12h, Subfunction 16h

*DESQview v2.20+*

### *SEND MESSAGE - "GETFLAGS" - GET OBJECT FLAGS*

**Registers at call:**
AH = 12h
BH = 16h
BL = handle identifier (see Table 44-5)
mailbox, keyboard, or pointer only
STACK: DWORD object's handle (if BL even)

**Return Registers:**
STACK: DWORD current control flags (see Table 44-30)

**Details:** This function is only available if the API level has been set to at least 2.20.

If a mailbox has bits 4 or 5 set, you must use READINTO rather than READ (see Function 12h Subfunction 13h) to retrieve messages.

**See also:** Function 12h Subfunctions 0Ah, 0Bh, 13h ("READINTO"), and 15h

## INT 15h Function 12h, Subfunction 17h

*DESQview v2.42-2.52*

### *BUG*

**Registers at call:**
AH = 12h
BH = 17h
BL = handle identifier (see Table 44-5)
mailbox, keyboard, or pointer only
STACK: DWORD object's handle (if BL even)

**Return Registers:**
STACK popped if parameter passed on stack

**Details:** Due to a fencepost error, message 17h is accepted for mailboxes, keyboards, and pointers, but causes a random branch.

## INT 15h Function 12h, Subfunctions 80h to 96h

*DESQview v2.50+*

### *SEND MESSAGE WITH ERROR RECOVERY*

**Registers at call:**
AH = 12h
BH = 80h to 96h
other registers as appropriate

**Return Registers:**
AL = function status (see Table 44-31)
others as appropriate for function

**Details:** These functions are identical to Function 12h Subfunctions 00h through 16h, except that DESQview will not pop up a "Programming Error" window, instead returning an error code in AL (see Table 44-31).

**See also:** Function 12h Subfunctions 00h to 16h

**44-31 Error codes:**

| | |
|---|---|
| 00h | no error |
| 01h | invalid values |
| 02h | alias invalid |
| 03h | handle valid but wrong type |
| 04h | invalid handle |

---

### INT 15h Function DE00h

*DESQview*

***GET PROGRAM NAME***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE00h | AX = offset into DESQVIEW.DVO of program most recently selected from the "Switch Windows" menu (see Table 44-32) |

**Details:** This function always returns AX=0000h under DESQview/X.

**See also:** Function DE07h

**44-32 Format of program entry in DESQVIEW.DVO:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | length of name (FFh if end of file) |
| 01h | N BYTEs | name |
| | 2 BYTEs | keys to invoke program (second = 00h if only one key used) |
| | BYTE | program type |
| | | 00h normal program |
| | | 04h divider |
| | | 80h Delete a Program |
| | | 81h Change a Program |
| | WORD | *apparently always 0000h* |

---

### INT 15h Function DE01h

*DESQview*

***UPDATE "OPEN WINDOW" MENU***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE01h | none |

**Details:** This function reads DESQVIEW.DVO, and disables Open menu if the file is not in the current directory. It is a NOP for DESQview/X.

---

### INT 15h Function DE02h

*DESQview 1.x only*

***SET*** *UNKNOWN* ***FLAG FOR CURRENT WINDOW***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE02h | none |

**Details:** This call is a NOP in DV 2.x.

**See also:** Function DE03h

---

### INT 15h Function DE03h

*DESQview 1.x only*

***GET*** *UNKNOWN* ***FOR CURRENT WINDOW***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE03h | AX = *unknown* for current window<br>BX = *unknown* for current window |

**Details:** This call is a NOP in DV 2.x.

**See also:** Function DE02h

### INT 15h Function DE04h

*DESQview*

### *GET AVAILABLE COMMON MEMORY*

**Registers at call:**
AX = DE04h

**Return Registers:**
BX = bytes of common memory available
CX = largest block available
DX = total common memory in bytes

**See also:** Functions DE05h and DE06h

### INT 15h Function DE05h

*DESQview*

### *GET AVAILABLE CONVENTIONAL MEMORY*

**Registers at call:**
AX = DE05h

**Return Registers:**
BX = KB of memory available
CX = largest block available
DX = total conventional memory in KB

**See also:** Functions DE04h and DE06h

### INT 15h Function DE06h

*DESQview*

### *GET AVAILABLE EXPANDED MEMORY*

**Registers at call:**
AX = DE06h

**Return Registers:**
BX = KB of expanded memory available
CX = largest block available
DX = total expanded memory in KB

**See also:** Functions DE04h and DE05h

### INT 15h Function DE07h

*DESQview*

### *"APPNUM" - GET CURRENT PROGRAM'S NUMBER*

**Registers at call:**
AX = DE07h

**Return Registers:**
AX = number of program as it appears on the "Switch Windows" menu

**Details:** This API call may be made from a hardware interrupt handler.

**See also:** Function DE00h

### INT 15h Function DE08h

*DESQview*

### ***GET** UNKNOWN FLAG*

**Registers at call:**
AX = DE08h

**Return Registers:**
AX = result
0000h if *unknown* is not set to the current task
0001h if *unknown* is set to the current task

### INT 15h Function DE09h

*DESQview*

### *UNIMPLEMENTED*

**Registers at call:**
AX = DE09h

**Return Registers:**
none (NOP in DV 1.x and 2.x)

### INT 15h Function DE0Ah

*DESQview v2.00+*

### *"DBGPOKE" - DISPLAY CHARACTER ON STATUS LINE*

**Registers at call:**
AX = DE0Ah
BL = character

**Return Registers:**
character displayed, next call will display in next position (which wraps back to the start of the line if off the right edge of screen)

**Details:** This function displays the specified character on the 25th line of the *physical* screen, regardless of the current size of window (even entirely hidden). It does not know about graphics display modes, it just pokes the characters into display memory.

This API call may be made from a hardware interrupt handler.

**See also:** Function 1003h

### INT 15h Function DE0Bh

*DESQview v2.00+*

### *"APILEVEL" - DEFINE MINIMUM API LEVEL REQUIRED*

**Registers at call:**
AX = DE0Bh
BL = API level minor version number
BH = API level major version number

**Return Registers:**
AX = maximum API level (AH = major, AL = minor)

**Details:** If the requested API level is greater than the version of DESQview, a "You need a newer version" error window is popped up.

The API level defaults to 1.00, and is inherited by child tasks.

### INT 15h Function DE0Ch

*DESQview v2.00+*

### *"GETMEM" - ALLOCATE "SYSTEM" MEMORY*

**Registers at call:**
AX = DE0Ch
BX = number of bytes

**Return Registers:**
ES:DI -> allocated block or 0000h:0000h (DV 2.26+)

**Details:** Use SETERROR (Function DE15h) to avoid a user prompt if there is insufficient system memory.

**See also:** Functions 1001h, 102Eh, DE0Dh, DE15h, and DE19h

### INT 15h Function DE0Dh

*DESQview v2.00+*

### *"PUTMEM" - DEALLOCATE "SYSTEM" MEMORY*

**Registers at call:**
AX = DE0Dh
ES:DI -> previously allocated block

**Return Registers:**
none

**See also:** Functions 1002h and DE0Ch

### INT 15h Function DE0Eh

*DESQview v2.00+*

### *"FINDMAIL" - FIND MAILBOX BY NAME*

**Registers at call:**
AX = DE0Eh
ES:DI -> name to find
CX = length of name

**Return Registers:**
BX = status
  0000h not found
  0001h found
    DS:SI = object handle

**See also:** Function 12h Subfunctions 11h and 1200h (GETNAME)

**44-33 Special mailbox names:**

| | |
|---|---|
| "COM1" ... "COM4" | RBcomm using COM1 ... COM4 |
| "DESQview/X Help Engine" | |
| "DESQview/X Network Server" | Network Manager |
| "DESQview X Server0" | X-Windows server |
| "DESQview X Server7" | X-Windows printing service |
| "INBOX" | DESQview/X LPD requests |
| "OUTBOX" | DESQview/X LPD responses |
| "WAITBOX" | semaphore to synchronize DESQview/X LPD communications |
| "_DVNM_" | DV/X v1.10 network manager |

### INT 15h Function DE0Fh

*DESQview v2.00+*

### *ENABLE DESQview EXTENSIONS*

**Registers at call:**
AX = DE0Fh

**Return Registers:**
AX and BX destroyed (seems to be bug, weren't saved/restored)

**Details:** This function sends a manager stream with opcodes AEh, BDh, and BFh to task's window. It also enables an additional mouse mode.

---

### INT 15h Function DE10h

*DESQview v2.00+*

### *"PUSHKEY" - PUT KEY INTO KEYBOARD INPUT STREAM*

**Registers at call:**
AX = DE10h
BH = scan code
BL = character

**Return Registers:**
none

**Details:** A later read will get the keystroke as if it had been typed by the user. Multiple pushes are read last-in first-out.

If a script exists for the pushed key in the current application, the script will be executed.

Early copies of DV 2.00 destroy AX, BX, ES, and DI.

**See also:** INT 16h Function 05h

---

### INT 15h Function DE11h, Subfunction 00h

*DESQview v2.00+*

### *"JUSTIFY" - EN/DISABLE AUTOMATIC WINDOW JUSTIFICATION*

**Registers at call:**
AX = DE11h
BL = new state
  00h viewport will not move automatically
  nonzero viewport will move to keep cursor visible (default)

**Return Registers:**
none

---

### INT 15h Function DE12h, Subfunction 0000h

*DESQview v2.01+*

### *"CSTYLE" - SET "C"-COMPATIBLE CONTROL CHAR INTERPRETATION*

**Registers at call:**
AX = DE12h
BX = new state
  0000h select normal style (linefeed only moves down)
  nonzero select C style (linefeed moves to start of next line)

**Return Registers:**
none

**Details:** C style is set on a per-task basis, and is inherited from the parent task.

---

### INT 15h Function DE13h

*DESQview v2.20+*

### *"GETCRIT" - GET CRITICAL NESTING COUNT*

**Registers at call:**
AX = DE13h

**Return Registers:**
BX = number of calls to BEGINC or ENTERC (see INT 15h Functions 101Bh and DE1Ch) without matching ENDC (see INT 15h Function 101Ch)

**Details:** This API call may be made from within a hardware interrupt handler.

**See also:** Functions 101Bh, 101Ch, DE1Bh, and DE1Ch

---

### INT 15h Function DE14h

*DESQview v2.20+*

### *GET OBJECT TYPE*

**Registers at call:**
AX = DE14h
ES:DI -> object

**Return Registers:**
BL = result
  00h not an object
  08h window or task
  09h mailbox
  0Ah keyboard
  0Bh timer
  0Ch objectq
  0Fh pointer
  10h panel

**See also:** Function 1016h

---

### INT 15h Function DE15h

*DESQview v2.20+*

***"SETERROR" - SET ERROR HANDLING MODE***

**Registers at call:**
AX = DE15h
BL = new error handling mode
  00h post system error on all error conditions
  01h return carry flag set on calls to ADDTO, SUBFROM, and WRITE messages sent to mailboxes which fail due to lack of system or common memory
  02h (v2.26+) same as 01h, but also return null pointer for GETMEM calls which fail due to lack of system memory

**Return Registers:**
none

**See also:** Functions DE0Ch and DE16h

---

### INT 15h Function DE16h

*DESQview v2.20+*

***"GETERROR" GET ERROR HANDLING MODE***

**Registers at call:**
AX = DE16h

**Return Registers:**
BL = current mode (see Function DE15h)

**See also:** Function DE15h

---

### INT 15h Function DE17h

*DESQview v2.20-2.25*

***reserved***

**Registers at call:**
AX = DE17h

**Return Registers:**
pops up "Programming error" window

**Details:** AX = 1117h is NOT identical to this call under DESQview 2.20 thru 2.25.

**See also:** Function 1117h

---

### INT 15h Function DE17h

*DESQview v2.26+*

***"ASSERTMAP" - GET/SET MAPPING CONTEXT***

**Registers at call:**
AX = DE17h
BX = function
  0000h get current mapping context without setting
  nonzero set new mapping context to BX

**Return Registers:**
BX = mapping context in effect before call

**Details:** Mapping contexts determine conventional-memory addressability; setting a mapping context ensures that the associated program and data areas are in memory for access. This function is usable by drivers, TSRs and shared programs; the caller need not be running under DESQview.

This API call may be made from a hardware interrupt handler.

**See also:** Functions 1016h, 1117h, and DE21h; MS Windows INT 2Fh Function 1685h (Chapter 43)

---

### INT 15h Function DE18h

*DESQview v2.20+*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = DE18h
BP = function number
  high byte must be 10h
  low byte is function
    00h set *unknown*
      BL = *unknown* (00h-10h, *video mode*)
      BH = value to store
    03h set *unknown*
      BL = *unknown* (stored in driver)
    0Ah get *unknown*
      ES:DI -> 18-byte buffer to hold *unknown*

**Return Registers:**
none

**Details:** This function calls the video driver (NOP for Hercules driver,probably CGA and MCGA also).

---

### INT 15h Function DE19h

*DESQview v2.23+*

### *"GETCOMMON" - ALLOCATE "COMMON" MEMORY*

**Registers at call:**
AX = DE19h
BX = number of bytes to allocate

**Return Registers:**
AX = status
  0000h successful
    ES:DI -> allocated block
  nonzero insufficient memory

**Details:** This API call may be made from within a hardware interrupt handler.
**See also:** Functions DE0Ch, DE15h, and DE1Ah

---

### INT 15h Function DE1Ah

*DESQview v2.23+*

### *"PUTCOMMON" - DEALLOCATE "COMMON" MEMORY*

**Registers at call:**
AX = DE1Ah
ES:DI -> previously allocated block

**Return Registers:**
AX = 0000h (successful)

**Details:** This function may be called from within a hardware interrupt handler.
**See also:** Functions DE0Dh and DE19h

---

### INT 15h Function DE1Bh

*DESQview v2.23+ internal*

### *DECREMENT CRITICAL NESTING COUNT*

**Registers at call:**
AX = DE1Bh

**Return Registers:**
none

**See also:** Functions 101Ch, DE13h, and DE1Ch

---

### INT 15h Function DE1Ch

*DESQview v2.23+*

### *"ENTERC" - INCREMENT CRITICAL NESTING COUNT*

**Registers at call:**
AX = DE1Ch

**Return Registers:**
none

**Details:** Similar to Function 101Bh, but begins the critical region without ensuring that DOS is free.

The official documentation states that this call should be paired with "ENDC" (Function 101Ch); no mention is made of Function DE1Bh.

This API call may be made from within a hardware interrupt handler.

**See also:** Functions 101Bh, 101Ch, DE13h, and DE1Bh

---

### INT 15h Function DE1Dh

*DESQview v2.23+*

### *"PUTKEY" - FAKE USER KEYSTROKES*

**Registers at call:**
AX = DE1Dh
DX = segment of handle for task to receive keystroke
BL = character
BH = scan code

**Return Registers:**
AX = status
  0000h if successful
  nonzero if receiver's keyboard buffer was full

**Details:** The key is treated as though the user had pressed it, ignoring any script which may be bound to the key, and using the current field table if the keyboard object is in field processing mode.

Multiple PUTKEYs are seen in the order in which they are executed.

**See also:** Function DE10h

---

### INT 15h Function DE1Eh

*DESQview v2.23+*

### *"SCRNINFO" - GET TRUE VIDEO PARAMETERS*

**Registers at call:**
AX = DE1Eh

**Return Registers:**
CL = actual number of rows on screen

CH = actual number of columns on screen
BL = actual video mode (may differ from INT 10h Function 0Fh return) (v2.26+)

**Details:** This API call may be made from a hardware interrupt handler.
**See also:** BIOS INT 10h Function 0Fh (Chapter 8)

---

### INT 15h Function DE1Fh

*DESQview v2.23+*

### *"DOSUSER" - GET HANDLE OF TASK CURRENTLY USING DOS*

**Registers at call:**
AX = DE1Fh

**Return Registers:**
BX = segment of task handle or 0000h if no tasks are using DOS

**Details:** This API call may be made from within a hardware interrupt handler.
**See also:** Function DE13h, MS-DOS INT 21h Function 34h (Chapter 24)

---

### INT 15h Function DE20h

*DESQview v2.26+*

### *"DISPATCHINT" - INTERRUPT ANOTHER TASK*

**Registers at call:**
AX = DE20h
BX = segment of handle of task to interupt
DX:CX -> FAR interrupt routine
BP,SI,DI,DS,ES as required by interrupt routine

**Return Registers:**
none

**Details:** Unlike "PGMINT" (Function 1021h), DISPATCHINT may be applied to the task making the DISPATCHINT call. Multiple "DISPATCHINT" calls are processed in the order in which they were executed.

The FAR routine is entered with the current ES, DS, SI, DI, and BP values, using the task's internal stack (see Function 101Ah); only SS:SP needs to be preserved.

This API call may be made from within a hardware interrupt handler.
**See also:** Functions 1021h and DE2Ah

---

### INT 15h Function DE21h

*DESQview v2.26+*

### *"ASSERTVIR" - CONTROL 386 SCREEN VIRTUALIZATION*

**Registers at call:**
AX = DE21h
BX = new state
  0000h turn off
  nonzero turn on

**Return Registers:**
BX = old state of virtualization

**Details:** This API call may be made from within a hardware interrupt handler.

Under DV 2.40 and 2.42, this call appears to have no effect and always returns a nonzero value in BX which appears to be the offset within the DV common memory segment of the caller's task object; it may only have an effect within a hardware interrupt handler.
**See also:** Functions 1117h and DE17h

---

### INT 15h Function DE22h

*DESQview v2.26+*

### *"PROCESSMEM" - GET TASK MEMORY STATUS*

**Registers at call:**
AX = DE22h
DX = segment of task handle

**Return Registers:**
DX = total amount of memory in paragraphs
BX = amount of system memory in paragraphs
CX = largest block of system memory available in paragraphs
AX = flags (see Table 44-34)

**Details:** If the task handle is a child task, the returned values will be for the process containing the task, rather than the task itself.

If the process's system memory is swapped out, BX,CX,DX remain unchanged, because the memory usage cannot be determined.
**See also:** Functions DE04h, DE05h, and DE06h

**44-34 Bitfields for process flags:**

| | |
|---|---|
| bit 0 | system memory resides in shared memory |
| bit 1 | process's memory is swapped out |
| bit 2 | process's system memory is swapped out |

### INT 15h Function DE23h

*DESQview v2.31+ (undocumented)*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = DE23h
BX = *unknown* IRQ number on first PIC?
CX = *unknown* IRQ number on second PIC?

**Return Registers:**
*unknown*

**Details:** This function is called by QEMM 6.00+.

### INT 15h Function DE25h

*DESQview v2.40+*

***"GETDVPATH" - GET DESQview DIRECTORY***

**Registers at call:**
AX = DE25h
ES:DI -> 67-byte buffer for ASCIZ directory name

**Return Registers:**
ES:DI buffer filled with directory from which DESQview was started

**BUG:** DESQview v2.42 does not place a terminating NUL at the end of the directory name, so if the buffer is not cleared to zeros before the call, there is no way to tell where the directory name ends. This bug has been fixed in DV 2.52 (DV/X 1.02)

**See also:** Function DE2Eh, MS-DOS INT 21h Function 47h (Chapter 24)

### INT 15h Function DE26h

*DESQview v2.40+*

***"GETFOREGROUND" - GET KEYBOARD FOCUS***

**Registers at call:**
AX = DE26h

**Return Registers:**
BX = segment of handle for task with keyboard focus

**Details:** Under DESQview/X, the X server always has the keyboard focus unless a "direct" window is active.

**See also:** Function DE2Fh, INT 2Fh Function DE0Ah

### INT 15h Function DE27h

*DESQview v2.50+*

***"ADDINSTANCEDATA" - ADD PER-TASK SAVE/RESTORE AREA***

**Registers at call:**
AX = DE27h
BX = type
  0000h process
  0001h task
ES:DI -> list of Instance Item Structures (see Table 44-35)

**Return Registers:**
CF clear if successful
  AX = *unknown*
  BX = *unknown*
CF set on error
  AX = *error code*
    0004h invalid BX value

**See also:** INT 2Fh Functions DE08h and DE09h

**44-35 Format of Instance Item Structure [one element of list]:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | length of data area DESQview should save and restore on context switches (0000h = end of list) |
| 02h | DWORD | pointer to area to be saved/restored |

### INT 15h Function DE28h

*DESQview v2.50+ (undocumented)*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = DE28h

BX = segment of *unknown* or 0000h for default
*unknown*

**Return Registers:**
*unknown*

**See also:** Function DE2Ah

---

### INT 15h Function DE2Ah

*DESQview v2.50+*

### *"DISPATCHINTAFTERDOS" - INTERRUPT ANOTHER TASK*

**Registers at call:**
AX = DE2Ah
BX = segment of handle for task to interrupt or 0000h for caller
DX:CX -> interrupt routine
BP,SI,DI,DS,ES as required by interrupt routine

**Return Registers:**
none

**Details:** This call is the same as Function DE20h except that it will delay interrupting the specified task until after it has exited DOS.

**See also:** Functions 1021h and DE20h

---

### INT 15h Function DE2Bh

*DESQview v2.50+*

### *"OBJNEXT" - TRAVERSE OBJECT LIST*

**Registers at call:**
AX = DE2Bh
ES:DI -> starting object
    0000h:0000h for *first object in list*

**Return Registers:**
AX = status
    0000h successful
        ES:DI -> next object of same type
          (window/non-window)
    0001h failed (ES:DI was not a valid handle)

**Details:** There are two separate lists, one for window/task objects and one for all other objects.

**See also:** Functions 1016h, DE2Ah, and DE2Ch

---

### INT 15h Function DE2Ch

*DESQview v2.50+*

### *"WININFO" - GET WINDOW INFORMATION*

**Registers at call:**
AX = DE2Ch
DX = window information format version (0100h for DESQview 2.5x)
BX = segment of window handle or 0000h for default
ES:DI -> buffer for window information (see Table 44-36)

**Return Registers:**
AX = status
    0000h successful

**See also:** Functions 1000h, 1016h, DE01h, and DE2Bh

**44-36 Format of window information:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | task flag: 00h window, 01h task |
| 01h | BYTE | process number if owner task<br>00h if non-owner task |
| 02h | WORD | segment of owner's handle, 0000h if orphaned |
| 04h | WORD | mapping context (see Function 1016h) |
| 06h | BYTE | task status (see Table 44-37) |
| 07h | BYTE | unused |
| 08h | WORD | status bits (see Table 44-38) |
| 0Ah | BYTE | 01h if foreground-only window |

**44-37 Values for task status:**

| | |
|---|---|
| 00h | "Waiting" waiting for input |
| 01h | "Idle" keyboard poll limit reached |
| 03h | same as 01h |
| 04h | "Pausing" INT 15h Function 1000h pause called |

| | |
|---|---|
| 04h | DV/X direct: user did something to allow task switch |
| 05h | “ModeChg” video mode about to be changed |
| 06h | “ModeNtf” notify that video mode changed |
| 07h | “MoniCh” requested change to other monitor |
| 08h | “StartPgm” control relinquished to start new process |
| 09h | “MgrCan” made window manager CANCEL command |
| 0Ah | “Slicing” time slice expired |
| 0Bh | “Exit DOS” notify on DOS calls |
| 0Ch | “Enter DOS” process is re-entering DOS |
| 0Dh | “Terminate” INT 21h Function 4Ch or task freed |
| 0Eh | “BrkNxt” Control-Break pressed |
| 0Fh | “MgrCol” keyboard focus taken away |
| 10h | “PgmInt” interrupted by API call from another task |
| 11h | “BldOpen” call to INT 15h Function DE01h |

### 44-38 Bitfields for status bits:

| | |
|---|---|
| bit 0 | DESQview process |
| bit 1 | process swapped out |
| bit 2 | process is resized direct window (suspended) |
| bit 3 | process suspended itself |
| bit 4 | user suspended process |
| bit 5 | process is being created |
| bit 6 | task is freeing another task |

---

## INT 15h Function DE2Fh *DESQview v2.50+*

### *"VIDEONOTIFY" - HAS DIRECT WINDOW BEEN ACTIVE?*

| Registers at call: | Return Registers: |
|---|---|
| AX = DE2Fh | BX = status<br>0001h keyboard focus has been given to a direct window since the last call<br>0000h if not |

**Details:** Quarterdeck stated that this call would not be available under future versions of DESQview Classic; however, it is present in DV 2.60.

---

## INT 21h Function 2Bh, Subfunction 4445h *DESQview*

### *INSTALLATION CHECK*

**Registers at call:**
AH = 2Bh
CX = 4445h ('DE')
DX = 5351h ('SQ')
AL = subfunction (DV v2.00+)
    01h get version

    02h get shadow buffer info and start shadowing

    04h get shadow buffer info

**Return Registers:**
AL = FFh if DESQview not installed

05h stop shadowing

BX = version (BH = major, BL = minor)

**Details:** Early copies of v2.00 return 0002h.
BH = rows in shadow buffer
BL = columns in shadow buffer
DX = segment of shadow buffer

BH = rows in shadow buffer
BL = columns in shadow buffer
DX = segment of shadow buffer

**Details:** In DESQview v1.x, there were no subfunctions; this call only identified whether or not DESQview was loaded. DESQview v2.52 performs function 01h for all subfunction requests 0Ch and higher and appears to ignore all lower-numbered functions not listed here.

**BUG:** subfunction 05h does not appear to work correctly in DESQview 2.52.

**See also:** INT 15h Function DE30h, TopView INT 10h Functions FEh and FFh (above), TopView INT 15h Function 1024h (above)

---

### INT 50h to INT 57h
*DESQview*
### *RELOCATED IRQ0 to IRQ7*

**Details:** This is the default location for older versions; DESQview v2.26+ searches for unused ranges of interrupts and uses the lowest available range in its list for relocating these IRQs and the next lowest for relocating IRQ8-IRQ15.

A range of eight interrupts starting at a multiple of 8 is considered available if all vectors are identical and it has not been excluded with an /XB:nn commandline switch.

The list of ranges for v2.26 is 50h,58h,68h,78h,F8h (if fewer than two of these are available, F8h and then 50h are used anyway).

The list of ranges for v2.31+ is 68h,78h,88h-B8h,F8h (if fewer than two of these are available, F8h and then F0h are used anyway).

**Conflicts:** OS/2 v1.x (Chapter 26), Vanderaart Text Windows (Chapter 66), PC Thuis Organizer shell (Chapter 66), TIL Expert AIM (NI), IBM 3278 Control Program (NI), WEB (NI)

**See also:** INT 58h, IRQ0 INT 08h (Chapter 3), Screen Thief INT D8h (Chapter 66)

---

### INT 58h to INT 5Fh
*DESQview v2.26+*
### *RELOCATED IRQ8 to IRQ15*

**Details:** This is the default, but other INTs may be used (see DESQview INT 50h).

**Conflicts:** GSS*CGI (Chapter 8), HP 95LX graphics primitives (Chapter 20), DoubleDOS (Chapter 49), SitBack (Chapter 66), Alloy NTNX (NI), ATALK.SYS (NI), Cluster Adapter (NI), IBM 802.2 interface (NI), ISOLAN (NI), Microsoft Network Transport Layer (NI), NetBIOS (NI), TOPS (NI), $25 LAN (NI)

**See also:** INT 50h, IRQ8 INT 70h (Chapter 3)

---

### INT 67h Function 6Bh
*DESQview 2.42-2.53*
### *BUG*

**Registers at call:**
AH = 6Bh

**Return Registers:**
crash

**BUG:** The EMM.DVR portion of DESQview branches to a random location on this function due to a fencepost error.

## DESQview/X

DESQview/X is a fusion of the DESQview multitasker, an X Windows server (NCD's PC-Xview), and a DOS extender (Rational Systems' DOS/4GX). DESQview v2.50-2.52 are distributed as part of DESQview/X v1.0x; DESQview v2.53 is distributed as part of DESQview/X v1.10.

---

**INT 15h Function BFDEh, Subfunction 0000h** *DESQview/X*

***DVDOS4GX.DVR - INSTALLATION CHECK***

**Registers at call:**
AX = BFDEh
BX = 0000h

**Return Registers:**
AX = *unknown* (0003h)
BX = FFFFh

**See also:** Function BF02h (Chapter 35)

---

**INT 15h Function BFDEh, Subfunction 0001h** *DESQview/X 1.02+*

***DVDOS4GX.DVR - GET PROCESS MANAGER NAME***

**Registers at call:**
AX = BFDEh
BX = 0001h

**Return Registers:**
BX = 0000h (success)
CX:DX -> name of process manager executable

**See also:** Function BFDEh Subfunction 0000h

---

**INT 15h Function BFDEh, Subfunctions 0002h and 0003h** *DESQview/X 1.02+*

***DVDOS4GX.DVR - GET/SET*** *UNKNOWN POINTER*

**Registers at call:**
AX = BFDEh
BX = subfunction
    0002h set pointer
        CX:DX -> *unknown*
    0003h get pointer

**Return Registers:**
BX = 0000h (success)
CX:DX -> *unknown* (0003h get pointer)

**See also:** Function BFDEh Subfunction 0000h

---

**INT 15h Function BFDEh, Subfunction 0004h** *DESQview/X 1.02+*

***DVDOS4GX.DVR*** *- UNDETERMINED FUNCTION*

**Registers at call:**
AX = BFDEh
BX = 0004h
CL = *unknown*

**Return Registers:**
BX = 0000h (success)
CX:DX -> XBRK structure (see Table 35-1)

**See also:** Function BFDEh Subfunction 0000h

---

**INT 15h Function BFDEh, Subfunction 0005h** *DESQview/X 1.02+*

***DVDOS4GX.DVR*** *- UNDETERMINED FUNCTION*

**Registers at call:**
AX = BFDEh
BX = 0005h
CX = new value for *unknown*

**Return Registers:**
BX = 0000h (success)
AX = old value of *unknown*
DS:SI -> *unknown* (if AX nonzero on return)
ES:DI -> *unknown* (if AX zero on return)

**Details:** This function is called by DOS4GW.EXE.

**See also:** Function BFDEh Subfuction 0000h

---

**INT 15h Function BFDEh, Subfunctions 0006h and 0007h** *DESQview/X 1.02+*

***DVDOS4GX.DVR - GET/SET*** *UNKNOWN VALUES*

**Registers at call:**
AX = BFDEh
BX = subfunction

0006h get

0007h set
CX:DX = *unknown*

**Return Registers:**
BX = 0000h (success)

AH = *interrupt number* (BEh)
CX:DX = *unknown*

**See also:** Function BFDEh Subfunction 0000h, INT BEh

---

### INT 15h Function BFDEh, Subfunction 0008h

*DESQview/X 1.02+*

***DVDOS4GX.DVR*** *- UNDETERMINED FUNCTION*

**Registers at call:**
AX = BFDEh
BX = 0008h
CX = segment of *unknown*
DS = *unknown*

**Return Registers:**
BX = status
  0000h successful
    AL = *unknown* (80h or C0h)
    DX = *unknown* (0603h) if AL=C0h
  0001h failed
    AX = 0000h

**Details:** This function is called by DOS4GW.EXE.
**See also:** Function BFDEh Subfunction 0000h

---

### INT 15h Function BFDEh, Subfunction 0009h

*DESQview/X 1.02+*

***DVDOS4GX.DVR - GET PROTECTED MODE PROGRAM LOADER***

**Registers at call:**
AX = BFDEh
BX = 0009h

**Return Registers:**
BX = 0000h (success)
CX:DX -> full pathname to LOAD32.EXP

**See also:** Function BFDEh Subfunction 0000h

---

### INT 15h Function BFDEh, Subfunctions 000Ah and 000Bh

*DESQview/X 1.02+*

***DVDOS4GX.DVR - INCREMENT/DECREMENT*** *UNKNOWN COUNTER*

**Registers at call:**
AX = BFDEh
BX = subfunction
  000Ah decrement
  000Bh increment

**Return Registers:**
BX = 0000h (success)
AX = new value of *unknown* counter

**Details:** Subfunction 000Ah also resets a variety of values if the counter goes negative.
These functions are called by DOS4GW.EXE.
**See also:** Function BFDEh Subfunction 0000h

---

### INT 15h Function BFDEh, Subfunction 000Ch

*DESQview/X 1.02+*

***DVDOS4GX.DVR*** *- UNDETERMINED FUNCTION*

**Registers at call:**
AX = BFDEh
BX = 000Ch
CL = *unknown*
  00h
  nonzero

**Return Registers:**
*unknown*

**See also:** Function BFDEh Subfunction 0000h

---

### INT 15h Function BFDEh, Subfunction 000Dh

*DESQview/X 1.02+*

***DVDOS4GX.DVR*** *- UNDETERMINED FUNCTION*

**Registers at call:**
AX = BFDEh
BX = 000Dh
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function BFDEh Subfunction 0000h

**INT 15h Function BFDEh, Subfunctions 000Eh and 000Fh** *DESQview/X 1.02+*

***DVDOS4GX.DVR*** *- UNDETERMINED FUNCTIONS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = BFDEh | AX = segment of handle for calling task |
| BX = subfunction | BX = *unknown* (probably destroyed) |
| | DX:CX -> *unknown* |

000Eh *unknown function*
    DX:CX -> *unknown*
000Fh *unknown function*

**Details:** Subfunction 000Fh is identical to subfunction 000Eh with CX:DX set to 0000h:0000h.

**See also:** Function BFDEh Subfunctions 0000h, 0010h, and 0013h

**INT 15h Function BFDEh, Subfunction 0010h** *DESQview/X 1.02+*

***DVDOS4GX.DVR - GET TASK HANDLE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = BFDEh | AX = segment of caller's task handle |
| BX = 0010h | BX destroyed |

**See also:** Function BFDEh Subfunction 000Fh

**INT 15h Function BFDEh, Subfunction 0011h** *DESQview/X 1.02+*

***DVDOS4GX.DVR - GET*** *UNKNOWN VALUES*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = BFDEh | CX = code segment of DVDOS4GX.DVR |
| BX = 0011h | BX = *unknown* (0004h) |

**See also:** Function BFDEh Subfunction 0000h

**INT 15h Function BFDEh, Subfunction 0012h** *DESQview/X 1.02+*

***DVDOS4GX.DVR - GET*** *UNKNOWN VALUES*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = BFDEh | DX = code segment of DVDOS4GX.DVR |
| BX = 0012h | BX = *unknown* (012Ch) |
| | CX = *unknown* (0006h) |

**See also:** Function BFDEh Subfunction 0000h

**INT 15h Function BFDEh, Subfunction 0013h** *DESQview/X 1.02+*

***DVDOS4GX.DVR - GET*** *UNKNOWN POINTER*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = BFDEh | DX:CX -> *unknown* |
| BX = 0013h | |

**See also:** Function BFDEh Subfunction 000Eh

**INT 15h Function BFDEh, Subfunctions 0014h and 0015h** *DESQview/X 1.02+*

***DVDOS4GX.DVR - LOCK/UNLOCK*** *UNKNOWN* ***MAILBOX***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = BFDEh | AX,BX destroyed |
| BX = subfunction | |

    0014h lock
    0015h unlock

CX = index of *unknown* mailbox (0000h-0004h valid, but no range checking done)

**See also:** Function BFDEh Subfunctios 0016h and 0017h

### INT 15h Function BFDEh, Subfunction 0016h

*DESQview/X 1.02+*

***DVDOS4GX.DVR - CHECK IF** UNKNOWN **MAILBOX OWNED***

**Registers at call:**
AX = BFDEh
BX = 0016h
CX = index of *unknown* mailbox (0000h-0004h valid, but no range checking done)

**Return Registers:**
AX = status
0000h no one owns mailbox
0001h mailbox has an owner
BX destroyed

**See also:** Function BFDEh Subfunctions 0015h and 0017h

### INT 15h Function BFDEh, Subfunction 0017h

*DESQview/X 1.02+*

***DVDOS4GX.DVR - GET** UNKNOWN **MAILBOX OWNER***

**Registers at call:**
AX = BFDEh
BX = 0017h
CX = index of *unknown* mailbox (0000h-0004h valid, but no range checking done)

**Return Registers:**
AX = segment of mailbox owner's handle
BX = segment of caller's task handle

**See also:** Function BFDEh Subfunctions 0015h and 0016h

### INT 15h Function BFDEh, Subfunctions FFFDh and FFFEh

*DESQview/X 1.02+*

***DVDOS4GX.DVR - GET/SET** UNKNOWN VALUE*

**Registers at call:**
AX = BFDEh
BX = subfunction
FFFDh get
FFFEh set
CX:DX = *unknown*

**Return Registers:**
(FFFDh) CX:DX = *unknown*

### INT 15h Function BFDEh, Subfunction FFFFh

*DESQview/X 1.02+*

***DVDOS4GX.DVR - NOP***

**Registers at call:**
AX = BFDEh
BX = FFFFh

**Return Registers:**
none

**See also:** Function BFDEh Subfunction 0000h

### INT 15h Function DE24h

*DESQview v2.40+*

***"XNEWPROC" - START NEW APPLICATION***

**Registers at call:**
AX = DE24h
BX = length of .DVP data
CX = length of *unknown* string
DS:SI -> *unknown* string
ES:DI -> .DVP data (see Function 102Ch)

**Return Registers:**
BX = segment of *task handle* or 0000h on error

**Details:** This call is similar to Function 102Ch except that it can interpret the extended DVP data. Under regular DESQview (non-/X), this function always returns BX=0000h, though it does not signal a programming error.

**See also:** TopView Function 102Ch (above)

### INT 15h Function DE29h, Subfunction 0000h

*DESQview/X*
*(undocumented)*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = DE29h
BX = 0000h
*unknown*

**Return Registers:**
CF clear if successful
*unknown*
CF set on error

**Details:** Under DESQview v2.60, this function and all other subfunctions always returns CF set, as they are unique to DESQview/X.

### INT 15h Function DE29h, Subfunctions 0001h and 0002h

*DESQview/X*
*(undocumented)*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = DE29h
BX = 0001h and 0002h
DX = segment of window handle

**Return Registers:**
CF clear if successful
AX = *unknown*
DX = *unknown*
CF set on error

### INT 15h Function DE29h, Subfunction 0003h

*DESQview/X*
*(undocumented)*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = DE29h
BX = 0003h
DX = segment of window handle

**Return Registers:**
CF clear if successful
*unknown*
CF set on error

### INT 15h Function DE29h, Subfunction 0004h

*DESQview/X*
*(undocumented)*

***GET DISPLAY NAME***

**Registers at call:**
AX = DE29h
BX = 0004h
CX = size of buffer in bytes
DX = segment of window handle
ES:DI -> buffer for display name

**Return Registers:**
CF clear if successful
buffer filled with ASCIZ display name (truncated if necessary) or null string if no display
CF set on error

**Details:** The name ":0" refers to the local display.

### INT 15h Function DE29h, Subfunction 0005h

*DESQview/X*
*(undocumented)*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = DE29h
BX = 0005h
*unknown*

**Return Registers:**
CF clear if successful
*unknown*
CF set on error

### INT 15h Function DE30h

*DESQview v2.50+*

***"GETDVXVERSION" - GET DESQview/X VERSION***

**Registers at call:**
AX = DE30h

**Return Registers:**
BX = version (BH=major, BL=minor) or 0000h if not DESQview/X

**Details:** You must first check the DESQview version to verify that it is 2.50 or greater.

**See also:** INT 21h Function 2Bh Subfunction 4445h

### INT 15h Function DE31h

*DESQview/X v1.10*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = DE31h
CX = *unknown*

0000h *unknown*
nonzero *unknown*
*unknown*

**Return Registers:**
*unknown*

---

**INT BEh** *DESQview/X*
*UNDETERMINED FUNCTION*
**Details:** This vector points at an IRET.
**See also:** INT 15h Function BFDEh Subfunction 0006h, DESQview INT 63h (NI)

## DESQview External Device Interface

The XDI permits programs (usually TSRs loaded before DESQview, but also DESQview shared programs and TSRs loaded inside a DESQview partition) to keep track of the state of the system under DESQview. With a few exceptions, the functions in this section are called by DESQview itself, and should not be called by applications.

Although there are no direct conflicts with the default multiplex number of DEh used by the XDI, there are numerous programs in the range C0h to FFh scanned by the XDI which potentially conflict; these are listed in Table 2-2.

---

**INT 2Fh Function DE00h** *DESQview v2.26+ XDI*
***INSTALLATION CHECK***
**Registers at call:**
AX = DE00h
BX = 4445h ("DE")
CX = 5844h ("XD")
DX = 4931h ("I1")

**Return Registers:**
AL = FFh if installed (even if other registers do not match)
if BX,CX, and DX were as specified on entry,
  BX = 4845h ("HE")
  CX = 5245h ("RE")
  DX = 4456h ("DV")

**Details:** AH=DEh is the default XDI multiplex number, but may range from C0h-FFh.

Programs should check for XDI starting at DEh to FFh, then C0h to DDh.

The XDI handler should not issue any DOS or BIOS calls, nor should it issue DESQview API calls other than those allowed from hardware interrupts.

**See also:** Function DE02h, OmniView INT 15h Function 5400h (Chapter 49)

---

**INT 2Fh Function DE01h** *DESQview v2.26+ XDI*
***DRIVER CUSTOM SUBFUNCTION***
**Registers at call:**
AX = DE01h
BX = driver ID
other registers as needed by driver

**Return Registers:**
vary

**Details:** XDI drivers should pass this call through to the previous handler if the ID does not match.

DESQview never calls this function; it is used exclusively for communication between XDI drivers and other applications. See the next section for several drivers' custom subfunctions.

---

**INT 2Fh Function DE02h** *DESQview v2.26+ XDI Callout*
***DV INITIALIZATION COMPLETE***
**Registers at call:**
AX = DE02h
BX = mapping context of DESQview (see Table 44-2)
DX = handle of DESQview system task

**Return Registers:**
none

**Details:** The driver should pass this call to the previous handler after doing its work.
**See also:** Functions DE03h and DE0Fh, OmniView INT 15h Function 5400h (Chapter 49)

---

**INT 2Fh Function DE03h** *DESQview v2.26+ XDI*
***DV TERMINATION*** *Callout*

**Registers at call:**
AX = DE03h
BX = mapping context of DESQview
DX = handle of DESQview system task

**Return Registers:**
none

**Details:** The driver should pass this call to the previous handler before doing its work.

DESQview makes this call when it is exiting, but before unhooking any interrupt vectors.

**See also:** Functions DE02h and DE0Fh, OmniView INT 15h Function 5407h (Chapter 49)

---

**INT 2Fh Function DE04h** *DESQview v2.26+ XDI*
***ADD PROCESS*** *Callout*

**Registers at call:**
AX = DE04h
BX = mapping context of new process (see Table 44-2)
DX = handle of process

**Return Registers:**
none

**Details:** The XMS XDI handler (installed by default) allocates a 22-byte record (see Table 44-39) from "common" memory to control access to XMS memory.

All DOS, BIOS, and DV API calls are valid inside the handler.

The driver should pass this call to the previous handler after processing it.

**See also:** Functions DE05h and DE06h, OmniView INT 15h Function 5401h (Chapter 49)

**44-39 Format of XMS XDI structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to *10-byte record* |
| 04h | DWORD | pointer to next XMS XDI structure |
| 08h | WORD | mapping context |
| 0Ah | BYTE | *unknown* |
| 0Bh | 5 BYTEs | XMS entry point to return for INT 2Fh Function 4310h (FAR jump to next field) |
| 10h | 6 BYTEs | FAR handler for XMS driver entry point (consists of a FAR CALL followed by RETF) |

---

**INT 2Fh Function DE05h** *DESQview v2.26+ XDI*
***REMOVE PROCESS*** *Callout*

**Registers at call:**
AX = DE05h
BX = mapping context of process (see Table 44-2)
DX = handle of last task in process

**Return Registers:**
none

**Details:** The XMS XDI handler releases the structure allocated by Function DE04h.

The driver should pass this call to the previous handler before processing it.

All DOS, BIOS, and DV API calls except those generating a task switch are valid inside the handler.

**See also:** Functions DE04h and DE07h, OmniView INT 15h Function 5402h (Chapter 49)

---

**INT 2Fh Function DE06h** *DESQview v2.26+ XDI*
***CREATE TASK*** *Callout*

**Registers at call:**
AX = DE06h
BX = mapping context of process containing task
DX = handle of new task

**Return Registers:**
none

**Details:** The driver should pass this call to the previous handler after processing it.

All DOS, BIOS, and DV API calls are valid inside handler.

**See also:** Function DE07h

**INT 2Fh Function DE07h** *DESQview v2.26+ XDI Callout*

***TERMINATE TASK***

| Registers at call: | Return Registers: |
|---|---|
| AX = DE07h | none |
| BX = mapping context of process containing task | |
| DX = handle of task | |

**Details:** The driver should pass this call to the previous handler before processing it.

All DOS, BIOS, and DV API calls except those generating a task switch are valid in handler.

**See also:** Functions DE04h, DE06h, and DE10h

---

**INT 2Fh Function DE08h** *DESQview v2.26+ XDI Callout*

***SAVE STATE***

| Registers at call: | Return Registers: |
|---|---|
| AX = DE08h | none |
| BX = mapping context of task being switched from (see Table 44-2) | |
| DX = handle of task being switched from | |

**Details:** This function is invoked prior to a task swap, interrupts, etc.

The driver should pass this call to the previous handler after processing it.

**See also:** Function DE09h, INT 15h Function DE27h (above), OmniView INT 15h Function 5403h (Chapter 49)

---

**INT 2Fh Function DE09h** *DESQview v2.26+ XDI Callout*

***RESTORE STATE***

| Registers at call: | Return Registers: |
|---|---|
| AX = DE09h | none |
| BX = mapping context of task being switched to (see Table 44-2) | |
| DX = handle of task being switched to | |

**Details:** The state is restored except for interrupts.

The driver should pass this call to the previous handler before processing it.

**See also:** Function DE08h, INT 15h Function DE27h (above), OmniView INT 15h Function 5404h (Chapter 49)

---

**INT 2Fh Function DE0Ah** *DESQview v2.26+ XDI Callout*

***CHANGE KEYBOARD FOCUS***

| Registers at call: | Return Registers: |
|---|---|
| AX = DE0Ah | none |
| BX = mapping context of task receiving focus | |
| DX = handle of running task | |

**Details:** The driver should pass this call to the previous handler before processing it.

This call often occurs inside a keyboard interrupt.

DV 2.42 does not provide this call to XDI handlers running inside a window; instead, it directly calls the INT 2Fh handler which was active at the time DV started.

**See also:** INT 15h Functions DE26h and DE2Fh (above)

---

**INT 2Fh Function DE0Bh** *DESQview v2.26+ XDI*
***DVP PROCESSING COMPLETE*** *Callout*

| Registers at call: | Return Registers: |
|---|---|
| AX = DE0Bh | CX incremented as needed |

BX = mapping context of DESQview system task
CX = number of system memory paragraphs required for the use of all XDI drivers (DV will add this to system memory in DVP buffer)
DX = handle of DESQview system task
SI = mapping context of new process if it starts
ES:DI -> DVP buffer

**Details:** Once DV invokes this function, the DVP buffer contents may be changed. The driver should pass this call to the previous handler before processing it.

**See also:** Function DE0Eh

---

**INT 2Fh Function DE0Ch** *DESQview v2.26+ XDI*
***SWAP OUT PROCESS*** *Callout*

| Registers at call: | Return Registers: |
|---|---|
| AX = DE0Ch | none |

BX = mapping context of task being swapped out (see Table 44-2)
DX = handle of DESQview system task

**Details:** The driver should pass this call to the previous handler after processing it.

**See also:** Function DE0Dh

---

**INT 2Fh Function DE0Dh** *DESQview v2.26+ XDI*
***SWAP IN PROCESS*** *Callout*

| Registers at call: | Return Registers: |
|---|---|
| AX = DE0Dh | none |

BX = mapping context of process just swapped in (see Table 44-2)
DX = handle of DESQview system task

**Details:** The driver should pass this call to the previous handler before processing it.

**See also:** Function DE0Ch

---

**INT 2Fh Function DE0Eh** *DESQview v2.26+ XDI*
***DVP START FAILED*** *Callout*

| Registers at call: | Return Registers: |
|---|---|
| AX = DE0Eh | none |

BX = mapping context of DESQview system task
DX = handle of DESQview system task
SI = mapping context of failed process (same as for call to Function DE0Bh)

**Details:** The driver should pass this call to the previous handler after processing it.

**See also:** Function DE0Bh

---

**INT 2Fh Function DE0Fh** *DESQview v2.50+ XDI*
***INITIALIZE DV*** *Callout*

| Registers at call: | Return Registers: |
|---|---|
| AX = DE0Fh | none |

**Details:** DESQview 2.50+ calls this function just before it completes its initialization. At the time of the call, DESQview has not yet changed any interrupt vectors.

**See also:** Function DE02h

---

### INT 2Fh Function DE10h
*DESQview v2.50+ XDI Callout*

***FREE TASK***

**Registers at call:**
AX = DE10h
BX = mapping context of process (see Table 44-2)
DX = task handle of process

**Return Registers:**
none

**Details:** DESQview 2.50+ calls this function before it frees the task; it is similar to Function DE07h but allows the XDI handler to make calls which cause context switches.

**See also:** Functions DE06h and DE07h

## Miscellaneous

This final section covers a number of calls related to TopView and DESQview which are not part of the products themselves.

---

### INT 15h Function 11DEh
*QEXT.SYS*

***INSTALLATION CHECK***

**Purpose:** QEXT.SYS is Quarterdeck's HMA manager for DESQview; more recent versions also implement the XMS standard.

**Registers at call:**
AX = 11DEh

**Return Registers:**
CF clear if installed
AX = segment at which QEXT.SYS is located

**Details:** A private entry point may be found by searching the beginning of the returned segment for the signature string "QUARTERDECK EXTENDED MEMORY MANAGER 286"; the word immediately prior to the signature contains the QEXT version number in BCD, and the word prior to that contains the offset within the QEXT code segment of the private entry point (see Table 44-40)

**See also:** XMS INT 2Fh Function 4310h (Chapter 38), QEMM INT 67h Function 3Fh (Chapter 39)

**44-40 Private entry point calling convention:**

**Registers at call:**
AH = 00h *unknown*
AH = nonzero *unknown*

**Return Registers:**
none

---

### INT 21h Function 30FFh, Subfunction 4445h
*DESQ*

***INSTALLATION CHECK***

**Registers at call:**
AX = 30FFh
CX = 4445h ("DE")
DX = 5351h ("SQ")

**Return Registers:**
BH = 05h if installed
*unknown*

**Details:** This function is called by DUBLDISK.COM v2.6; it is not supported by DESQview, so it may be for DESQview's precursor DESQ.

**See also:** DUBLDISK Function 4404h (Chapter 16)

---

### INT 2Fh Function DE01h, Subfunction 5242h
*DESQview v2.26+ XDI*

***CUSTOM SUBFUNCTION, Ralf Brown's XDI drivers***

**Registers at call:**
AX = DE01h
BX = 5242h ("RB")
CX:DX = program identifier
656F7000h ("eop",0) for DVeop

**Return Registers:**
AX = 5242h ("RB") if installed
ES:BX -> data or entry point (see Table 44-41)
CX = version number (CH = major, CL = minor)

**44-41 Call DVeop entry point with:**

| Registers at call: | Return Registers: |
|---|---|
| ES:DI -> callback address or<br>0000h:0000h to remove callback | AX = status<br>0000h failed (callback table full or attempted to remove non-existent callback)<br>0001h successful<br>ES:DI -> chaining address |
| | BX,CX,DX destroyed |

**Details:** The DVeop callback function is called with a simulated interrupt when the DESQview window containing it is closed; it should perform all necessary cleanup and then perform a FAR jump to the chaining address or an IRET if the chaining address is 0000h:0000h.

If the program wishes to remove itself before the window is closed, it should call the DVeop entry point with the previously returned chaining address and ignore the returned chaining address.

### INT 2Fh Function DE01h, Subfunction 7474h

*DESQview v2.26+ XDI*

***CUSTOM SUBFUNCTION, DVTXDI.COM***

| Registers at call: | Return Registers: |
|---|---|
| AX = DE01h | BX = 4F4Bh ("OK") |
| BX = 7474h | DL destroyed |
| CL = function | |
| 00h installation check | AL = FFh if installed |
| 01h get process handle<br>DX = keys on Open Window menu (DL = first, DH = second) | AX = process handle or 0000h if not running |
| 02h (v1.3+) set TMAN handle<br>DX = TMAN process handle | |
| 03h (v1.3+) set open keys to ignore on next CL=01h call<br>DX = keys on Open Window menu (DL = first, DH = second) | |

**Program:** DVTXDI is distributed as part of the shareware products DVTree (DOS Shell/DESQview process manager) and DVTMAN by Mike Weaver

### INT 2Fh Function DE01h, Subfunction 7575h

*DESQview v2.26+ XDI*

***CUSTOM SUBFUNCTION, DVSIXDI.COM***

**Registers at call:**

AX = DE01h
BX = 7575h
CX = subfunction
- 0000h installation check
- 0001h turn on notification (currently unused)
- 0002h turn off notification (currently unused)
- 0003h get process information

**Return Registers:**

0004h get version

AX = 00FFh if installed

0005h (v1.10+) get time since DESQview started

AX = 0001h

AX = 0001h

0006h (v1.10+) get number of task switches

AX = status
  0000h failed
  0001h successful
    BX = last instantaneous time slice in 1/100s (v1.10) or in 1/18s (v1.11+)
    CX = number of processes
    DX = number of "(starting)" records (v2.00+)
    SI = number of records in process info array (v2.00+) (always 15 for v1.x)
    ES:DI -> process info array (see Table 44-42 and 44-43)

AH = major version
AL = minor version

DX:AX = 1/100s since DV start (v1.10)
DX:AX = 1/18s since DV start (v1.11+)

DX:AX = total task switches
CX = task switches in last instantaneous interval

**Details:** DVSIXDI is part of the DVSI (DESQview System Information) package by Daniel J. Bodoh.

For v1.00, subfunction 0003h allocates common memory, which the caller must deallocate after reading the process information; only the currently used records are placed in the buffer. For v1.10+, function 0003h merely returns a pointer to the internal array of process information; the caller should make a copy of the array while inside a critical section (see INT 15h Function 101Bh). Only those records with bit 7 of the first byte set are valid.

**44-42 Format of information for one process (v1.00):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | flags<br>bit 7: process slot is valid |
| 01h | WORD | offset into DESQVIEW.DVO of program's record if started from Open Windows menu, else undefined |
| 03h | WORD | Switch Windows window number |
| 05h | WORD | segment of process handle |
| 07h | WORD | number of tasks owned by process |
| 09h | WORD | mapping context of process (see Table 44-2) |
| 0Bh | DWORD | hook for other programs |

## INT 2Fh Function DE10h

### 44-43 Format of information for one process (v1.10-v2.00):

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | flags<br>bit 7: valid record<br>bit 6: (v2.00+) record is allocated; if bit 7 clear, process is "(starting)" and only offsets 01h and 09h are valid<br>bit 5: (v2.00+) this app currently owns the CPU<br>bit 4: reserved (0)<br>bit 3: DESQview system task<br>bit 2: reserved (0)<br>bit 1: task has keyboard (currently unused)<br>bit 0: task swapped out (currently unused) |
| 01h | WORD | Open Window keys |
| 03h | WORD | Switch Windows number |
| 05h | WORD | segment of process handle |
| 07h | WORD | number of tasks for process |
| 09h | WORD | process mapping context |
| 0Bh | DWORD | time process started (relative to start of DESQview) |
| 0Fh | DWORD | time process last got CPU (relative to start of DESQview) |
| 13h | DWORD | time process last gave up CPU (relative to start of DESQview) |
| 17h | DWORD | total CPU time since process started |
| 1Bh | DWORD | CPU time at start of current instantaneous interval |
| 1Fh | DWORD | CPU time in current instantaneous interval |
| 23h | DWORD | hook for other programs |

**Note:** all times are in 1/100s for v1.10, in 1/18s for v1.11+

---

### INT 2Fh Function DE01h, Subfunction FFFEh

*DESQview v2.26+ XDI*

***DVXMS.DVR*** *- UNDETERMINED FUNCTION*

*(undocumented)*

**Registers at call:**
AX = DE01h
BX = FFFEh
CX = 4D47h ("MG")
DX = 0052h (0,"R")

**Return Registers:**
AL = FFh
DX = 584Dh ("XM")

# CHAPTER 45

# *MultiDOS Plus*

Nanosoft's MultiDOS Plus is a multitasker for MS-DOS applications. The two separate sections of this chapter cover the multitasker proper and the I/O drivers.

## Multitasker

---

### INT 15h Function 00h
*MultiDOS Plus*

#### *GIVE UP TIME SLICE*

| Registers at call: | Return Registers: |
|---|---|
| AH = 00h | none |

**Details:** If this function is issued by the highest-priority task while MultiDOS is using priority-based rather than round-robin scheduling, control will be returned to the caller immediately.
**Conflicts:** Cassette (Chapter 4), Amstrad PC1512 mouse (Chapter 21), VMiX (Chapter 46)
**See also:** Function 03h, TopView Function 1000h (Chapter 44)

---

### INT 15h Function 01h
*MultiDOS Plus*

#### *REQUEST RESOURCE SEMAPHORE*

| Registers at call: | Return Registers: |
|---|---|
| AH = 01h | AH = status (00h,02h) (see Table 45-1) |
| AL = semaphore number (00h-3Fh) | |

**Conflicts:** Cassette (Chapter 4), Amstrad PC1512 (Chapter 7), VMiX (Chapter 46)

**45-1 Values for function status:**

| | |
|---|---|
| 00h | successful |
| 01h | not semaphore owner |
| 02h | invalid semaphore number |
| 03h | semaphore already owned by caller |
| 04h | out of string space |

**Details:** If the semaphore is not owned, ownership is assigned to the calling task and the call returns immediately. If the semaphore is already owned by another task, the calling task is placed on a queue for the semaphore and suspended until it can become owner of the semaphore.

Semaphore 0 is used internally by MultiDOS to synchronize DOS access.

**See also:** Functions 02h, 10h, and 1Bh

---

### INT 15h Function 02h
*MultiDOS Plus*

#### *RELEASE RESOURCE SEMAPHORE*

| Registers at call: | Return Registers: |
|---|---|
| AH = 02h | AH = status (00h-02h) (see Table 45-1) |
| AL = semaphore number (00h-3Fh) | |

**Details:** If any tasks are waiting for the semaphore, the first task on the wait queue will become the new owner and be reawakened.

Do not use this function within an interrupt handler.

**Conflicts:** Cassette (Chapter 4), Amstrad PC1512 (Chapter 7), VMiX (Chapter 46)
**See also:** Functions 01h, 10h, and 1Ch

---

### INT 15h Function 03h

*MultiDOS Plus*

***SUSPEND TASK FOR INTERVAL***

**Registers at call:**
AH = 03h
DX = number of time slices to remain suspended

**Return Registers:**
after specified interval has elapsed

**Details:** When priority-based scheduling is in use, high-priority tasks should use this function to yield the processor.
**Conflicts:** Cassette (Chapter 4), Amstrad PC1512 (Chapter 7), VMiX (Chapter 46)
**See also:** Functions 00h and 0Ah

---

### INT 15h Function 04h

*MultiDOS Plus*

***SEND MESSAGE TO ANOTHER TASK***

**Registers at call:**
AH = 04h
AL = mailbox number (00h-3Fh)
CX = message length in bytes
DS:SI -> message

**Return Registers:**
AH = status (00h-02h) (see Table 45-2)

**Details:** The message is copied into a system buffer; the caller may immediately reuse its buffer.
**Conflicts:** ABIOS (Chapter 4), Amstrad PC1512 (Chapter 7), VMiX (Chapter 46)
**See also:** Function 05h

**45-2 Values for function status:**

| | |
|---|---|
| 00h | successful |
| 01h | out of message memory |
| 02h | invalid mailbox number |

---

### INT 15h Function 05h

*MultiDOS Plus*

***CHECK MAILBOX***

**Registers at call:**
AH = 05h
AL = mailbox number (00h-3Fh)

**Return Registers:**
AH = status (00h,02h) (see Table 45-2)
DX = (if AH=00h) length of first message in queue, 0000h if no message

**Conflicts:** ABIOS (Chapter 4), Amstrad PC1512 (Chapter 7), VMiX (Chapter 46)
**See also:** Functions 04h and 06h

---

### INT 15h Function 06h

*MultiDOS Plus*

***READ MAILBOX***

**Registers at call:**
AH = 06h
AL = mailbox number (00h-3Fh)
CX = size of buffer in bytes
ES:DI -> buffer for message

**Return Registers:**
AH = status (00h,02h) (see also Table 45-2)
  00h successful
    CX = number of bytes copied
    DX = actual length of message

**Details:** If the caller's buffer is not large enough, the message is truncated and the remainder is lost.
**Conflicts:** Amstrad PC1512 (Chapter 7), VMiX (Chapter 46)
**See also:** Functions 04h and 05h

---

## INT 15h Function 07h

*MultiDOS Plus*

### SPAWN INTERNAL TASK (CREATE NEW THREAD)

**Registers at call:**
AH = 07h
BX:CX = entry point of new task
DX = stack size in paragraphs

**Return Registers:**
AH = status
  00h successful
  01h no free task control blocks
  02h no free memory for task's stack

**Details:** Execution returns immediately to the calling task.
**Conflicts:** VMiX (Chapter 46)
**See also:** Functions 08h, 09h, and 13h

---

## INT 15h Function 08h

*MultiDOS Plus*

### TERMINATE INTERNAL TASK (KILL THREAD)

**Registers at call:**
AH = 08h

**Return Registers:**
calling task is terminated, so execution never returns to caller

**Details:** An internal task must be terminated with this function rather than a DOS termination function.

The task's stack space is returned to the parent task's memory pool.

**Conflicts:** VMiX (Chapter 46)
**See also:** Function 07h

---

## INT 15h Function 09h

*MultiDOS Plus*

### CHANGE TASK'S PRIORITY

**Registers at call:**
AH = 09h
AL = new priority

**Return Registers:**
none

**Details:** The priority has different meanings depending on whether priority-based or round-robin scheduling is used.
**Conflicts:** VMiX (Chapter 46)
**See also:** Function 07h

---

## INT 15h Function 0Ah

*MultiDOS Plus*

### CHANGE TIME SLICE INTERVAL

**Registers at call:**
AH = 0Ah
AL = new interval
  00h = 55.0 ms (default)
  80h = 27.5 ms
  40h = 13.75 ms
  20h = 6.88 ms
  10h = 3.44 ms
  08h = 1.72 ms

**Return Registers:**
none

**Conflicts:** VMiX (Chapter 46)
**See also:** Function 03h

---

## INT 15h Function 0Bh

*MultiDOS Plus*

### FORCE DISPLAY OUTPUT TO PHYSICAL SCREEN MEMORY

**Registers at call:**
AH = 0Bh

**Return Registers:**
none

**Details:** This function sets the calling task's screen pointer to the actual screen memory; the pointer may be restored with Function 0Ch.

Caller's video mode must be same as foreground task's video mode.

Any text written while in the background will be saved to the foreground task's virtual screen when it switches to the background.

This function is useful if a background task wants to display a message on the foreground screen.

**Conflicts:** VMiX (Chapter 46)
**See also:** Function 0Ch

---

## INT 15h Function 0Ch

*MultiDOS Plus*

### *RESTORE OLD VIDEO DISPLAY MEMORY*

| Registers at call: | Return Registers: |
|---|---|
| AH = 0Ch | none |

**Details:** This function restores the task's screen pointer saved by Function 0Bh; it must not be called unless Function 0Bh has been called first.

**Conflicts:** VMiX (Chapter 46)
**See also:** Function 0Bh

---

## INT 15h Function 0Dh

*MultiDOS Plus*

### *DISABLE MULTITASKING*

| Registers at call: | Return Registers: |
|---|---|
| AH = 0Dh | none |

**Details:** The calling task receives all time slices until Function 0Eh is called; this allows time-critical events or nonreentrant code to be processed.

**Conflicts:** VMiX (Chapter 46)
**See also:** Functions 0Eh, 10h, and 20h; TopView Function 101Bh (Chapter 44)

---

## INT 15h Function 0Eh

*MultiDOS Plus*

### *ENABLE MULTITASKING*

| Registers at call: | Return Registers: |
|---|---|
| AH = 0Eh | none |

**Conflicts:** VMiX (Chapter 46)
**See also:** Functions 0Dh and 20h, TopView Function 101Ch (Chapter 44)

---

## INT 15h Function 0Fh

*MultiDOS Plus*

### *EXECUTE A MULTIDOS PLUS COMMAND*

| Registers at call: | Return Registers: |
|---|---|
| AH = 0Fh | after command has been processed |
| DS:BX -> ASCIZ command | |

**Details:** The specified string is executed as if it had been typed at the MultiDOS command prompt. The task is placed on a queue which MultiDOS examines periodically and is suspended until MultiDOS has processed the command.

All lowercase characters up to the first blank are converted to upper case within the given buffer.

**Conflicts:** ESDI BIOS (Chapter 10), VMiX (Chapter 46)

---

## INT 15h Function 10h

*MultiDOS Plus*

### *TEST RESOURCE SEMAPHORE*

| Registers at call: | Return Registers: |
|---|---|
| AH = 10h | AH = status (see also Table 45-1) |
| AL = semaphore number (00h-3Fh) | 00h semaphore not in use |

**Conflicts:** TopView (Chapter 44), VMiX (Chapter 46)
**See also:** Functions 02h, 0Dh, and 1Dh

---

**INT 15h Function 11h** *MultiDOS Plus*

***TURN OFF AltZ TOGGLE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 11h | none |

**Details:** This function disables the Alt-Z MultiDOS command/program-selection hotkey.

**Conflicts:** TopView (Chapter 44), VMiX (Chapter 46)

**See also:** Function 12h

---

**INT 15h Function 12h** *MultiDOS Plus*

***TURN ON AltZ TOGGLE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 12h | none |

**Details:** This function enables the Alt-Z MultiDOS command/program-selection hotkey.

**Conflicts:** TopView (Chapter 44), VMiX (Chapter 46)

**See also:** Function 11h

---

**INT 15h Function 13h** *MultiDOS Plus*

***GET TASK CONTROL BLOCK***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 13h | BX:AX -> task control block (see Table 45-3) |

**Conflicts:** VMiX (Chapter 46)

**See also:** Function 15h

**45-3 Format of MultiDOS Plus v4.0 task control block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to next TCB |
| 04h | 8 BYTEs | ASCIZ task name |
| 0Ch | 2 BYTEs | *unknown* |
| 0Eh | WORD | task PSP segment |
| 10h | WORD | abort/suspend flags |
| 12h | WORD | current screen segment (see Function 0Bh,Function 0Ch) |
| 14h | WORD | priority level (0000h-FFFEh) |
| 16h | WORD | time slice counter |
| 18h | 2 BYTEs | *unknown* |
| 1Ah | WORD | suspend timer value |
| 1Ch | WORD | stack segment |
| 1Eh | WORD | stack pointer |
| 20h | WORD | display type |
| 22h | WORD | display memory |
| 24h | 2 BYTEs | *unknown* |
| 26h | WORD | termination count |
| 28h | WORD | equipment flag for BIO10 driver |
| 2Ah | BYTE | background CRT mode |
| 2Bh | WORD | screen width in columns |
| 2Dh | WORD | screen size in bytes |
| 2Fh | WORD | segment of physical screen memory |
| 31h | 16 BYTEs | eight cursor positions |
| 41h | WORD | current cursor shape |
| 43h | BYTE | active display page |
| 44h | WORD | CRT controller I/O port base |
| 46h | 2 BYTEs | *unknown* |

| | | |
|---|---|---|
| 48h | WORD | foreground task flag |
| 4Ah | 6 BYTEs | *unknown* |
| 50h | WORD | saved video segment (see Function 0Bh,Function 0Ch) |
| 52h | DWORD | old INT 22h |
| 56h | DWORD | old INT 23h |
| 5Ah | DWORD | old INT 24h |
| 5Eh | WORD | top of memory for task |
| 60h | 4 BYTEs | *unknown* |
| 64h | WORD | DTA segment (see INT 21h Function 1Ah) |
| 66h | WORD | DTA offset |
| 68h | 4 BYTEs | *unknown* |
| 6Ch | BYTE | current ANSI.SYS attribute |
| 6Dh | BYTE | current ANSI.SYS column |
| 6Eh | BYTE | current ANSI.SYS row |
| 6Fh | BYTE | current ANSI.SYS display state |
| 70h | BYTE | maximum ANSI.SYS columns |
| 71h | BYTE | current ANSI.SYS page |
| 72h | WORD | saved ANSI.SYS cursor position |
| 74h | BYTE | ANSI.SYS parameter buffer index |
| 75h | BYTE | current ANSI.SYS screen mode |
| 76h | BYTE | ANSI.SYS wrap flag |
| 77h | 6 BYTEs | ANSI.SYS parameter buffer |
| 7Dh | BYTE | ANSI.SYS keyboard DSR state |
| 7Eh | 7 BYTEs | ANSI.SYS keyboard DSR buffer |
| 85h | 3 BYTEs | *unknown* |
| 88h | 16 BYTEs | request header for DOS driver calls |
| 98h | 14 BYTEs | *unknown* |
| A6h | WORD | segment of EMS map if EMS task |
| A8h | WORD | flag: task makes EMS calls |
| AAh | WORD | EMS handle for task |
| ACh | WORD | keyboard shift state |
| AEh | 12 BYTEs | *unknown* |
| BAh | WORD | TCB of parent if child task |
| BCh | WORD | termination code |
| BEh | WORD | COM port number |
| C0h | 4 BYTEs | *unknown* |
| C4h | WORD | current IRQ number |
| C6h | 2 BYTEs | *unknown* |
| C8h | WORD | miscellaneous flag word |
| CAh | 2 BYTEs | *unknown* |
| CCh | DWORD | old INT 10h |
| D0h | WORD | EMS alternate map set number |
| D2h | 414 BYTEs | DOS current disk and directory context (optional) |

## INT 15h Function 13h

## INT 15h Function 14h

*MultiDOS Plus*

### *CHECK IF MultiDOS FOREGROUND OR BACKGROUND*

**Registers at call:**
AH = 14h

**Return Registers:**
AX = current state
0000h MultiDOS Plus command prompt is background task
0001h command prompt is foreground task

**Conflicts:** VMiX (Chapter 46)
**See also:** Function 0Bh

## INT 15h Function 15h

*MultiDOS Plus*

### *GET SYSTEM BLOCK*

**Registers at call:**
AH = 15h

**Return Registers:**
BX:AX -> system block (see Table 45-4)

**Conflicts:** VMiX (Chapter 46)
**See also:** Function 13h

**45-4 Format of MultiDOS Plus 4.0 system block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of system control block |
| 02h | WORD | redirection flag set by /NOREDIRECT |
| 04h | WORD | no-INT 10h flag set by /NO10 |
| 06h | DWORD | old INT 10h |
| 0Ah | DWORD | new INT 10h |
| 0Eh | DWORD | pointer to WORD with current TCB offset (see Function 13h) |
| 12h | DWORD | pointer to WORD with idle task TCB offset |
| 16h | DWORD | pointer to WORD with foreground TCB offset |
| 1Ah | DWORD | pointer to WORD with MultiDOS TCB offset |
| 1Eh | WORD | Task Control Block size |
| 20h | WORD | number of TCBs |
| 22h | WORD | flag: EMS present |
| 24h | WORD | EMS page frame base segment |
| 26h | WORD | 16K pages in EMS page frame |
| 28h | WORD | base segment for conventional memory tasks |
| 2Ah | WORD | conventional memory size in paragraphs |
| 2Ch | DWORD | pointer to list of queue pointers |

## INT 15h Function 16h

*MultiDOS Plus*

### *INITIALIZATION*

**Registers at call:**
AH = 16h

**Return Registers:**
*unknown*

**Details:** This function is used internally during initialization; any other calls will cause unpredicatable results.

**Conflicts:** VMiX (Chapter 46)

## INT 15h Function 17h

*MultiDOS Plus*

### *MAP IRQ*

**Registers at call:**
AH = 17h
AL = IRQ to map (01h-0Fh)
BX = offset of task control block (see Table 45-3) to associate with IRQ

**Return Registers:**
AX = status
0000h successful
other invalid IRQ

**Details:** The EMS map of the specified TCB is associated with the given interrupt.
**Conflicts:** VMiX (Chapter 46)
**See also:** Functions 18h and 19h

---

### INT 15h Function 18h

*MultiDOS Plus*

***UNMAP IRQ***

| Registers at call: | Return Registers: |
|---|---|
| AH = 18h | AX = status |
| AL = IRQ to unmap (01h-0Fh) | 0000h successful |
| | 0001h invalid IRQ |

**Details:** Results are unpredictable if the IRQ has not been mapped.
**Conflicts:** VMiX (Chapter 46)
**See also:** Functions 17h and 19h

---

### INT 15h Function 19h

*MultiDOS Plus*

***UNMAP ALL IRQs***

| Registers at call: | Return Registers: |
|---|---|
| AH = 19h | AX destroyed |

**Details:** This function is for MultiDOS internal use only.
**Conflicts:** VMiX (Chapter 46)
**See also:** Functions 17h and 18h

---

### INT 15h Function 1Ah

*MultiDOS Plus*

***MAP SEMAPHORE NAME TO NUMBER***

| Registers at call: | Return Registers: |
|---|---|
| AH = 1Ah | AL = status (00h,04h) (see Table 45-1) |
| DS:SI -> 8-byte name | AH = (if AL=00h) semaphore number (20h-3Fh) |

**Details:** All eight bytes of the name are significant.

If the name does not already exist, it is added to the name table and associated with a free semaphore number.

Names cannot be destroyed.

**Conflicts:** VMiX (Chapter 46)
**See also:** Functions 1Bh, 1Ch, and 1Dh

---

### INT 15h Function 1Bh

*MultiDOS Plus*

***REQUEST RESOURCE SEMAPHORE BY NAME***

| Registers at call: | Return Registers: |
|---|---|
| AH = 1Bh | AH = status (00h,02h-04h) (see Table 45-1) |
| DS:SI -> 8-byte name | |

**Details:** This function is equivalent to Function 1Ah followed by Function 01h.
**Conflicts:** VMiX (Chapter 46)
**See also:** Functions 01h, 1Ah, 1Ch, and 1Dh

---

### INT 15h Function 1Ch

*MultiDOS Plus*

***RELEASE RESOURCE SEMAPHORE BY NAME***

| Registers at call: | Return Registers: |
|---|---|
| AH = 1Ch | AH = status (00h-02h,04h) (see Table 45-1) |
| DS:SI -> 8-byte name | |

**Details:** This function is equivalent to Function 1Ah followed by Function 02h.
**Conflicts:** VMiX (Chapter 46)
**See also:** Functions 02h, 1Ah, 1Bh, and 1Dh

### INT 15h Function 1Dh
*MultiDOS Plus*

### *TEST RESOURCE SEMAPHORE BY NAME*

| Registers at call: | Return Registers: |
|---|---|
| AH = 1Dh | AH = status (00h-04h) (see also Table 45-1) |
| DS:SI -> 8-byte name | 00h semaphore not in use |

**Details:** This function is equivalent to Function 1Ah followed by Function 10h.
**Conflicts:** VMiX (Chapter 46)
**See also:** Functions 10h, 1Ah, 1Bh, and 1Ch

### INT 15h Function 1E00h
*MultiDOS Plus*

### *CLEAR EVENT COUNTER*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1E00h | AH = status |
| DX = event/trigger number (00h-3Fh) | 00h successful |

**Conflicts:** VMiX (Chapter 46)
**See also:** Functions 1E01h and 1E02h

### INT 15h Function 1E01h
*MultiDOS Plus*

### *TRIGGER EVENT*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1E01h | AH = status |
| DX = event/trigger number (00h-3Fh) | 00h successful |
| | 01h invalid event/trigger number |

**Details:** This function schedules any task waiting for the event; if no task is waiting, the event counter is incremented (and will roll over if it was 65535). This function may be invoked by an interrupt handler.
**Conflicts:** VMiX (Chapter 46).
**See also:** Functions 1E00h and 1E02h

### INT 15h Function 1E02h
*MultiDOS Plus*

### *WAIT FOR EVENT*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1E02h | AH = status |
| DX = event/trigger number (00h-3Fh) | 00h successful |
| | 01h invalid event/trigger number |

**Details:** If the event counter is zero, the task is suspended until the event is triggered with Function 1E01h; else, the counter is decremented and the call returns immediately.
**Conflicts:** VMiX (Chapter 46)
**See also:** Functions 1E00h and 1E01h

### INT 15h Function 1E08h
*MultiDOS Plus 4.01*

### *SET CONTEXT-SWITCH FUNCTIONS*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1E08h | none |
| DX:BX -> context save handler | |
| DX:CX -> context restore handler | |

**Details:** The handlers may be removed by setting addresses to 0000h:0000h.
**Conflicts:** VMiX (Chapter 46)

**45-5 Context save/restore handler calling convention:**

**Registers at call:**
ES:BX -> task's TCB

**Return Registers:**
all registers preserved

---

### INT 15h Function 1Fh
### *GET MEMORY PARAMETERS*

*MultiDOS Plus v4.01*

**Registers at call:**
AH = 1Fh

**Return Registers:**
BX = first segment of conventional memory
DX = first segment of EMS swap frame into which MultiDOS will load programs

**Conflicts:** VMiX (Chapter 46)

### INT 15h Function 20h
### *CHECK IF MULTITASKING ENABLED*

*MultiDOS Plus v4.01*

**Registers at call:**
AH = 20h

**Return Registers:**
AX = current state
  0000h multitasking enabled
  other TCB of task that disabled multitasking

**Conflicts:** PRINT (Chapter 27)
**See also:** Functions 0Dh and 13h

## I/O Drivers

### INT 14h Function 04h
### *INITIALIZE PORT*

*MultiDOS Plus IODRV*

**Registers at call:**
AH = 04h

**Return Registers:**
port initialized; if Hayes-compatible modem, a connection has been established

**Details:** The port number is stored at offset BEh in the Task Control Block (see Table 45-3).
**Conflicts:** ROM BIOS (Chapter 12), FOSSIL (Chapter 12), DigiBoard (Chapter 12), MBBIOS (Chapter 12), Microsoft Systems Journal TSRCOMM (Chapter 12)
**See also:** Functions 05h and 20h, INT 15h Function 13h (above), BIOS INT 14h Function 00h (Chapter 12)

### INT 14h Function 05h
### *READ CHARACTER FROM PORT*

*MultiDOS Plus IODRV*

**Registers at call:**
AH = 05h
AL = timeout in seconds (00h = never)

**Return Registers:**
AL = status
  00h successful
    AH = character read
  01h read error
  02h timed out
  other modem status (CTS, DSR) changed

**Details:** The port number is stored at offset BEh in the Task Control Block.
**Conflicts:** ROM BIOS (Chapter 12), FOSSIL (Chapter 12), DigiBoard (Chapter 12), MBBIOS (Chapter 12), PC-MOS/386 (Chapter 49)
**See also:** Functions 04h, 06h, and 22h; INT 15h Function 13h (above); BIOS INT 14h Function 02h (Chapter 12)

### INT 14h Function 06h

*MultiDOS Plus IODRV*

#### *WRITE CHARACTER TO PORT*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 06h<br>AL = character | AL = status<br>00h successful |

**Details:** The port number is stored at offset BEh in the Task Control Block.

If the output queue is full, the calling task is blocked until the character can be stored.

**Conflicts:** FOSSIL (Chapter 12), DigiBoard (Chapter 12), MBBIOS (Chapter 12), PC-MOS/386 (Chapter 49), TelAPI (NI)

**See also:** Functions 04h, 05h, and 21h; INT 15h Function 13h (above); BIOS Function 01h (Chapter 12)

### INT 14h Function 07h

*MultiDOS Plus IODRV*

#### *GET PORT STATUS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 07h | CL = modem status (see Table 12-2)<br>CH = character at head of input queue (if any)<br>DX = number of characters in input queue |

**Details:** The port number is stored at offset BEh in the Task Control Block.

**Conflicts:** FOSSIL (Chapter 12), DigiBoard (Chapter 12), MBBIOS (Chapter 12), PC-MOS/386 (Chapter 49), TelAPI (NI)

**See also:** Functions 05h, 08h, 09h, and 23h; INT 15h Function 13h (above); BIOS Function 03h (Chapter 12)

### INT 14h Function 08h

*MultiDOS Plus 4.0 IODRV*

#### *GET AND RESET PORT LINE STATUS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 08h | AL = line status (see Table 12-2)<br>AH destroyed |

**Details:** The port number is stored at offset BEh in the Task Control Block.

On every line status change, the line status is ORed with the line status accumulator; this function returns the accumulator and clears it.

**Conflicts:** FOSSIL (Chapter 12), DigiBoard (Chapter 12), MBBIOS (Chapter 12), PC-MOS/386 (Chapter 49)

**See also:** Functions 04h and 07h, INT 15h Function 13h (above), BIOS INT 14h Function 03h (Chapter 12)

### INT 14h Function 09h

*MultiDOS Plus IODRV*

#### *RESET PORT STATUS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 09h | modem status byte cleared |

**Details:** The port number is stored at offset BEh in the Task Control Block. **Conflicts:** FOSSIL (Chapter 12), DigiBoard (Chapter 12), MBBIOS (Chapter 12), PC-MOS/386 (Chapter 49). **See also:** Functions 04h and 07h, INT 15h Function 13h (above)

### INT 14h Function 20h

*MultiDOS Plus*

#### *INITIALIZE PORT*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 20h<br>AL = port parameters (see Function 00h”SERIAL”)<br>DX = port number (0-3) | AH = status (00h,41h,64h) (see Table 45-6) |

**Conflicts:** FOSSIL (Chapter 12), PC-MOS/386 (Chapter 49), Alloy MW386 (NI)

**See also:** Functions 04h, 21h, and 23h; BIOS Function 00h (Chapter 12)

**45-6 Values for status:**

| | |
|---|---|
| 00h | successful |
| 39h | no DSR or CTS |
| 3Ah | invalid status storage specified |
| 3Bh | no CTS |
| 3Ch | no DSR |
| 3Dh | framing and parity error |
| 3Eh | overrun error |
| 3Fh | framing error |
| 40h | parity error |
| 41h | no such port |
| 42h | monitor mode not active |
| 64h | monitor mode already active |
| 96h | ring buffer overflow |
| 97h | timed out |

---

### INT 14h Function 21h
### *TRANSMIT CHARACTER*

*MultiDOS Plus*

**Registers at call:**
AH = 21h
AL = character to send
DX = port number

**Return Registers:**
AH = status (00h,39h,3Bh,3Ch,41h,42h,97h) (see Table 45-6)

**Details:** Monitor mode must have been turned on with Function 24h before calling this function.
**Conflicts:** FOSSIL (Chapter 12), PC-MOS/386 (Chapter 49), Alloy MW386 (NI)
**See also:** Functions 20h, 22h, and 24h

---

### INT 14h Function 22h
### *RECEIVE CHARACTER*

*MultiDOS Plus*

**Registers at call:**
AH = 22h
DX = port number

**Return Registers:**
AH = status (00h,3Dh-40h,96h) (see Table 45-6)
AL = character if successful

**Details:** If no character is available, this function waits until a character arrives or an implementation-dependent timeout elapses.
**Conflicts:** PC-MOS/386 (Chapter 49), Alloy MW386 (NI)
**See also:** Functions 20h, 21h, and 27h

---

### INT 14h Function 23h
### *GET PORT STATUS*

*MultiDOS Plus*

**Registers at call:**
AH = 23h
DX = port number

**Return Registers:**
AH = line status (see Function 03h)
AL = modem status (see Function 03h)

**Conflicts:** PC-MOS/386 (Chapter 49), Alloy MW386 (NI)
**See also:** Functions 07h and 20h, BIOS INT 14h Function 03h (Chapter 12)

---

### INT 14h Function 24h
### *SET MONITOR MODE*

*MultiDOS Plus*

**Registers at call:**
AH = 24h
AL = port status storage
  00h single status for entire receive buffer
  01h separate status kept for each byte in receive buffer
DX = port number

**Return Registers:**
AH = status (00h,3Ah,41h,64h) (see Table 45-6)

**Details:** In monitor mode, MultiDOS redirects all BIOS video output to a serial port.
**Conflicts:** PC-MOS/386 (Chapter 49), Alloy MW386 (NI)
**See also:** Functions 20h and 25h

---

### INT 14h Function 25h
*MultiDOS Plus*

### *CLEAR BUFFERS*

**Registers at call:**
AH = 25h
AL = function
    00h only clear buffers
    01h clear buffers and deactivate
DX = port number

**Return Registers:**
AH = status (00h,3Ah,41h,42h) (see also Table 45-6)
    3Ah invalid function

**See also:** Functions 20h and 24h

---

### INT 14h Function 27h
*MultiDOS Plus*

### *GET BUFFER CHARACTER COUNT*

**Registers at call:**
AH = 27h
DX = port number

**Return Registers:**
AH = status (00h,41h,42h) (see Table 45-6)
AL = number of characters in receive buffer

## CHAPTER 46

# *VMiX*

VMiX is a shareware multiuser multitasking system by Commercial Software Associates (ComSoft). Recent versions are able to run an 80386 or later CPU in Virtual-86 mode; all versions have been able to make use of 80286 protected mode for much of VMiX's operation.

---

### INT 15h Function 00h *VMiX v2+*

### *INSTALLATION CHECK*

**Registers at call:**
AH = 00h

**Return Registers:**
DX = 0798h if installed
AX = version (AH = major, AL = minor)

**Conflicts:** Cassette (Chapter 4), Amstrad PC1512 mouse (Chapter 21), MultiDOS Plus (Chapter 45)

---

### INT 15h Function 01h *VMiX*

### *"sys_chanreq" - I/O CHANNEL OBJECT MANAGER*

**Registers at call:**
AH = 01h
STACK contains parameters (see Table 46-1)

**Return Registers:**
DX:AX -> IRP structure or 0000h:0000h

**Conflicts:** Cassette (Chapter 4), Amstrad PC1512 (Chapter 7), MultiDOS Plus (Chapter 45)
**See also:** Functions 00h and 02h

**46-1 Channel Request parameters:**

| | | |
|---|---|---|
| *STACK:* | *WORD* | *object ID of requestor* |
| | DWORD | pointer to ASCIZ name of requested method |
| | "assign" | assign channel to object |
| | "deassign" | deassign channel |
| | "cursor" | set cursor on/off |
| | "init" | initialize comm port |
| | "open" | open I/O channel |
| | "position" | set cursor position |
| | "receive" | get buffered packet from comm port |
| | "send" | send buffered packet to comm port |
| | "vio" | set current virtual I/O to specified channel |
| | "window" | make window at cursor position |
| | **---if "assign"---** | |
| | WORD | object UID |
| | WORD | caller UID/PID |
| | DWORD | CSL with port |
| | **---if "deassign"---** | |
| | WORD | channel ID |

| | |
|---|---|
| ---if "cursor"--- | |
| WORD | channel ID (must be a SRCSINK) |
| WORD | new state (0000h off, 0001h on) |
| ---if "init"--- | |
| WORD | channel ID (must be a SRCSINK) |
| WORD | comm port number (00h-03h) |
| WORD | UART init code |
| ---if "open"--- | |
| WORD | channel ID |
| ---if "position"--- | |
| WORD | channel ID (must be a SRCSINK) |
| WORD | position (high byte = row, low byte = column) |
| ---if "receive"--- | |
| DWORD | pointer to buffer |
| ---if "send"--- | |
| WORD | length of buffer |
| DWORD | pointer to buffer |
| ---if "vio"--- | |
| WORD | channel ID (must be a SRCSINK) |
| ---if "window"--- | |
| WORD | top left (high byte = row, low byte = column) |
| WORD | bottom right (high byte = row, low byte = column) |

---

## INT 15h Function 02h *VMiX*

### *"sys_memreq" - MEMORY OBJECT MANAGER*

**Registers at call:**
AH = 02h
STACK contains parameters (see Table 46-2)

**Return Registers:**
DX:AX -> memory block or VPGE structure or 0000h:0000h

**Conflicts:** Cassette (Chapter 4), Amstrad PC1512 (Chapter 7), MultiDOS Plus (Chapter 45)
**See also:** Functions 00h and 01h

**46-2 Memory Manager parameters:**

| | | |
|---|---|---|
| *STACK:* | *WORD* | *object ID of requestor* |
| | DWORD | pointer to ASCIZ name of requested method |
| | "assign" | allocate low memory block |
| | "assign extended" | allocate extended memory pages |
| | "assign gdt" | allocate GDT selector |
| | "paged" | allocate low paged memory |
| | "paged extended" | alllocate extended memory pages |
| | "deassign" | free memory block |
| | "deassign gdt" | free GDT selector |
| | "getvpage" | get physical address for virtual page |
| | "setvpage" | set physical address for virtual page |
| | "info" | get VMiX memory usage info block |
| | "move" | move contents of 32-bit memory |
| | "newmcb" | make new DOS memory control block |
| | "owner" | get process ID of MCB or PSP owner |
| | "umb" | allocate upper memory block |
| | "video" | toggle system use of vidoe memory and get stat |
| | ---if "assign"--- | |
| | WORD | number of objects |

| | |
|---|---|
| WORD | size in bytes (multiple of 512 bytes) |
| **---if "assign extended"---** | |
| WORD | number of objects |
| WORD | size in bytes (multiple of 4K) |
| **---if "assign gdt"---** | |
| WORD | access type (low byte) |
| WORD | segment size in paragraphs |
| DWORD | pointer to start of physical segment |
| **---if "paged"---** | |
| WORD | number of 512-byte pages |
| **---if "paged extended"** | |
| WORD | number of 4K pages |
| **---if "deassign"---** | |
| DWORD | pointer returned by previous allocation call |
| **---if "deassign gdt"---** | |
| WORD | GDT selector |
| **---if "getvpage"---** | |
| WORD | owner's process ID |
| DWORD | pointer to buffer for page structure (struct VPGE) |
| **---if "setvpage"---** | |
| WORD | owner's process ID |
| DWORD | pointer to new page structure (struct VPGE) |
| **---if "info"---** | |
| no additional arguments | |
| **---if "move"---** | |
| DWORD | 32-bit source address |
| DWORD | 32-bit destination address |
| WORD number of words to move | |
| **---if "newmcb"---** | |
| DWORD | pointer to new MCB's location |
| WORD | size of memory block |
| DWORD | pointer to ASCIZ name string (max 8 chars) |
| **---if "owner"---** | |
| WORD | MCB or PSP segment |
| **---if "umb"---** | |
| WORD | size in paragraphs |
| **---if "video"---** | |
| no additional arguments | |

---

## INT 15h Function 03h *VMiX*

### *"sys_pinput" - PROMPTED CONSOLE INPUT*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 03h | AX = length of input (input buffer is padded with blanks) |
| STACK: DWORD pointer to ASCII prompt | |
| WORD field outline character | |
| WORD length of input field (max 7Fh) | |
| DWORD address of pointer to input buffer | |

**Conflicts:** Cassette (Chapter 4), Amstrad PC1512 (Chapter 7), MultiDOS Plus (Chapter 45)
**See also:** Function 04h

---

**INT 15h Function 04h** *VMiX*

***"sys_vprintf" - FORMATTED OUTPUT TO STREAM***

**Registers at call:**
AH = 04h
STACK: DWORD control string
DWORD array of arguments

**Return Registers:**
none

**Conflicts:** ABIOS (Chapter 4), Amstrad PC1512 (Chapter 7), MultiDOS Plus (Chapter 45)
**See also:** Function 03h

---

**INT 15h Function 05h** *VMiX*

***"sys_getpid" - GET PROCESS ID OF CURRENT PROCESS***

**Registers at call:**
AH = 05h

**Return Registers:**
AX = process ID

**Conflicts:** ABIOS (Chapter 4), Amstrad PC1512 (Chapter 7), MultiDOS Plus (Chapter 45)
**See also:** Functions 06h and 0Bh

---

**INT 15h Function 06h** *VMiX*

***"sys_getpcb" - GET POINTER TO PROCESS CONTROL BLOCK***

**Registers at call:**
AH = 06h
STACK: WORD process ID

**Return Registers:**
DX:AX -> process control block

**Conflicts:** Amstrad PC1512 (Chapter 7), MultiDOS Plus (Chapter 45)
**See also:** Functions 05h, 07h, and 08h

---

**INT 15h Function 07h** *VMiX*

***"sys_getocb" - GET POINTER TO OBJECT CONTROL BLOCK***

**Registers at call:**
AH = 07h
STACK: WORD object type

**Return Registers:**
DX:AX -> object control block

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Functions 06h and 08h

---

**INT 15h Function 08h** *VMiX*

***"sys_getccb" - GET CHANNEL CONTROL BLOCK***

**Registers at call:**
AH = 08h
STACK: WORD channel ID

**Return Registers:**
DX:AX -> channel control block

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Functions 06h and 07h

---

**INT 15h Function 09h** *VMiX*

***"sys_getqueue" - GET ID OF QUEUED ELEMENT***

**Registers at call:**
AH = 09h
STACK: WORD queue ID (see Table 46-3)
WORD subqueue ID

**Return Registers:**
AX = queue ID

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Function 0Ah

**46-3 Queue Identifiers:**

| | |
|---|---|
| 0000h | process queue |
| 0001h | object queue |
| 0003h | type queue |

## INT 15h Function 0Ah

*VMiX*

### *"sys_qetqnext" - GET ID OF NEXT QUEUED ELEMENT*

**Registers at call:**
AH = 0Ah
STACK: WORD queue ID (see Table 46-3)
WORD ID of current element in queue chain

**Return Registers:**
AX = ID of next element

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Functions 09h and 0Fh

## INT 15h Function 0Bh

*VMiX*

### *"sys_sysreq" - SYSTEM CONFIGURATION MANAGER*

**Registers at call:**
AH = 0Bh
STACK contains parameters (see Table 46-4)

**Return Registers:**
DX:AX -> result or 0000h:0000h
**---if "length"---**
BYTE receive offset
BYTE send offset
**---if "kswitch"---**
DWORD old stack pointer
**---if "numproc"---**
WORD number of active processes
**---if "status"---**
current open comm port status
**---if "xport"---**
current comm port being polled for logins

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Functions 05h and 0Eh

**46-4 System Configuration parameters:**

| | | |
|---|---|---|
| *STACK:* | *WORD* | *caller's UID* |
| | DWORD | pointer to ASCIZ name of requested method |
| | "abort" | abort current send/receive on comm port |
| | "block" | start/end critical section |
| | "close" | terminate interrupt-drive comm I/O |
| | "open" | prepare comm port for interrupt-driven I/O |
| | "delay" | set delay timer and wait |
| | "hibernate" | put process to sleep |
| | "ints" | enable/disable interrupt-driven INT 14h |
| | "length" | get current send/receive buffer offsets |
| | "kswitch" | switch stacks |
| | "numproc" | get number of active processes |
| | "protocol" | set protocol function for comm interrupts |
| | "relocate" | set/reset VMiX flag for relocating to himem |
| | "status" | get current open comm port status |
| | "wake" | awaken a process |
| | "xport" | get comm port polled for logins |

**---if "abort"---**
no additional arguments
**---if "block"---**
WORD 0000h end, 0001h start
**---if "close"---**
no additional arguments

---if "open"---
WORD comm port (00h-03h)
WORD BIOS parameter byte (see INT 14h Function 00h), except bits 7-5 = 000 for 19200, 001 for 38400, 011 for 115200
---if "delay"---
WORD time in seconds
---if "hibernate"---
WORD process ID
---if "ints"---
WORD 0000h if no, 0001h if yes
---if "length","numproc","relocate","status","xport"---
no additional arguments
---if "kswitch"---
DWORD pointer to new stack
---if "protocol"---
DWORD pointer to function (must be in low "assign"ed memory when in 386 mode)
---if "wake"---
WORD process ID

## INT 15h Function 0Ch

*VMiX*

### *"sys_getstack" - GET POINTER TO PROCESS TSS STACK*

**Registers at call:**
AH = 0Ch
STACK: WORD process ID

**Return Registers:**
DX:AX -> TSS stack store

**Conflicts:** MultiDOS Plus (Chapter 45). **See also:** Function 00

## INT 15h Function 0Dh

*VMiX*

### *"sys_spawn" - START A CHILD PROCESS JOB SHELL*

**Registers at call:**
AH = 0Dh
STACK: DWORD ASCIZ string starting with requested I/O channel and followed by standard VMiX shell command string

**Return Registers:**
AX = process ID or error code "SYS_ERROR"

**Details:** The maximum string length is 7Fh characters.
**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Functions 0Eh and 11h, MS-DOS INT 21h Function 4Bh (Chapter 24)

## INT 15h Function 0Eh

*VMiX*

### *"sys_kill" - HARD TERMINATE PROCESS*

**Registers at call:**
AH = 0Eh
STACK: WORD process ID

**Return Registers:**
AX = status (SYS_OK or SYS_ERROR)

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Functions 0Bh and 0Dh

## INT 15h Function 0Fh

*VMiX*

### *"sys_getqkey" - GET KEY FIELD OF QUEUED ELEMENT*

**Registers at call:**
AH = 0Fh
STACK: WORD queue ID (see Table 46-3)
WORD ID of element in queue chain

**Return Registers:**
AX = key

**Conflicts:** ESDI BIOS (Chapter 10), MultiDOS Plus (Chapter 45)
**See also:** Function 0Ah

**INT 15h Function 10h** *VMiX*

***"sys_virtual" - EXECUTE CONFORMING FUNCTION IN PROTECTED MODE***

| Registers at call: | Return Registers: |
|---|---|
| AH = 10h<br>STACK: DWORD pointer to function<br>N WORDsfunction args | AX = *function's return value* (not specified in documentation) |

**Details:** While the function is executing, the following global descriptors are available:

20h stack segment
38h code segment of function
40h data alias for function's code segment.

Additional GDT descriptors can be allocated using Function 02h with "assign gdt".

**Conflicts:** TopView (Chapter 44), MultiDOS Plus (Chapter 45)

**See also:** Functions 02h and 51h

**INT 15h Function 11h** *VMiX*

***"sys_system" - EXECUTE SHELL SYSTEM COMMANDS***

| Registers at call: | Return Registers: |
|---|---|
| AH = 11h | AX = status (SYS_OK or SYS_ERROR) |

STACK: DWORD pointer to ASCIZ string containing a VMiX shell request (max len = 127)

**Conflicts:** TopView (Chapter 44), MultiDOS Plus (Chapter 45)

**See also:** Function 0Ch

**INT 15h Function 12h** *VMiX*

***"sys_sleep" - PUT PROCESS TO SLEEP***

| Registers at call: | Return Registers: |
|---|---|
| AH = 12h<br>STACK: WORD process ID | AX = status (SYS_OK or SYS_ERROR) |

**Conflicts:** TopView (Chapter 44), MultiDOS Plus (Chapter 45)

**See also:** Function 13h, MultiDOS Function 03h (Chapter 45), TopView Function 101Dh (Chapter 44)

**INT 15h Function 13h** *VMiX*

***"sys_wake" - WAKE SLEEPING PROCESS***

| Registers at call: | Return Registers: |
|---|---|
| AH = 13h<br>STACK: WORD process ID | AX = status (SYS_OK or SYS_ERROR) |

**Conflicts:** MultiDOS Plus (Chapter 45)

**See also:** Function 12h

**INT 15h Function 14h** *VMiX*

***"sys_clrwindow" - CLEAR WINDOW***

| Registers at call: | Return Registers: |
|---|---|
| AH = 14h | AX = status (SYS_OK) |

STACK: WORD top left corner of window (high byte = row, low = col)
WORD bottom right corner of window (high = row, low = col)

**Details:** Clears window to color set with "sys_setcolors" (Function 18h).

**Conflicts:** MultiDOS Plus (Chapter 45)

**See also:** Functions 15h, 16h, 18h, and 1Fh, BIOS INT 10h Function 06h (Chapter 8)

---

### INT 15h Function 15h

VMiX

***"sys_setbwindow" - SET BANNER WINDOW MESSAGE***

**Registers at call:**
AH = 15h
STACK: DWORD pointer to ASCIZ banner message for top of screen

**Return Registers:**
AX = status (SYS_OK)

**Conflicts:** MultiDOS Plus (Chapter 45). **See also:** Functions 14h and 16h

---

### INT 15h Function 16h

VMiX

***"sys_setwindow" - SET ROOT WINDOW SIZE AND HOME CURSOR***

**Registers at call:**
AH = 16h
STACK: DWORD pointer to I/O Request Packet
WORD top left corner of window (high byte = row, low = col)
WORD bottom right corner of window (high = row, low = col)

**Return Registers:**
AX = status (SYS_OK or SYS_ERROR)

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Functions 14h and 17h

---

### INT 15h Function 17h

VMiX

***"sys_getcolors" - GET CONSOLE WINDOW COLORS***

**Registers at call:**
AH = 17h

**Return Registers:**
AH = foreground color
AL = background color

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Functions 16h and 18h

---

### INT 15h Function 18h

VMiX

***"sys_setcolors" - SET CONSOLE COLORS***

**Registers at call:**
AH = 18h
STACK: WORD new colors (see Table 46-5)

**Return Registers:**
AX = color

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Functions 14h, 17h, and 19h

**46-5 Bitfields for background/foreground colors:**

| | |
|---|---|
| bits 3-0 | foreground |
| bits 7-4 | background |
| bits 15-8 | unused |

---

### INT 15h Function 19h

VMiX v2+

***"sys_setconwn" - SET WINDOW COLORS***

**Registers at call:**
AH = 19h
STACK: WORD new colors (see Table 46-5)

**Return Registers:**
AX = color

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Function 18h

---

### INT 15h Function 1Ah

VMiX v2+

***"sys_sint" - INVOKE SOFTWARE INTERRUPT***

**Registers at call:**
AH = 1Ah
STACK: WORD interrupt number
DWORD pointer to register structure

**Return Registers:**
AX = returned flags

**Conflicts:** MultiDOS Plus (Chapter 45)

### INT 15h Function 1Bh

*VMiX v2+*

***"sys_blkmov" - MOVE MEMORY BLOCK***

**Registers at call:**
AH = 1Bh
STACK: DWORD source address
DWORD destination address
WORD number of words to move

**Return Registers:**
none

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** BIOS INT 15h Function 87h (Chapter 4)

### INT 15h Function 1Ch

*VMiX v2+*

***"sys_bitblt" - PUT GRAPHICAL OBJECT AT CURSOR POSITION***

**Registers at call:**
AH = 1Ch
STACK: WORD AND/OR pixel with background (00h OR, 01h AND)
DWORD pointer to object bitmap
WORD object width in pixels
WORD object height in pixels

**Return Registers:**
none

**Conflicts:** MultiDOS Plus (Chapter 45). **See also:** Function 1Dh

### INT 15h Function 1Dh

*VMiX v2+*

***"sys_getfont" - GET CURRENT CONSOLE GRAPHICS FONT***

**Registers at call:**
AH = 1Dh

**Return Registers:**
AX = current font number (00h-03h)

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Functions 1Ch and 1Eh

### INT 15h Function 1Eh

*VMiX v2+*

***"sys_setfont" - SET CONSOLE GRAPHICS FONT***

**Registers at call:**
AH = 1Eh
STACK: WORD new font number (00h-03h)

**Return Registers:**
AX = current font number (00h-03h)

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Function 1Dh

### INT 15h Function 1Fh

*VMiX v2.?+*

***"sys_scrollwin" - SCROLL WINDOW***

**Registers at call:**
AH = 1Fh
STACK: WORD top left corner of window (high byte = row, low = col)
WORD bottom right corner of window

**Return Registers:**
AX = status (SYS_OK)

**Conflicts:** MultiDOS Plus (Chapter 45)
**See also:** Functions 14h and 18h

### INT 15h Function 50h

*VMIX v2.?+*

***"sys_vm_page" - SET NEW VIRTUAL PAGE TABLE***

**Registers at call:**
AH = 50h
BX = segment of page directory table
CX = page number of page table

**Return Registers:**
none

**See also:** Functions 10h, 51h, and 52h

---

**INT 15h Function 51h** *VMiX v2.?+*

***"sys_vm_func" - EXECUTE FUNCTION IN PROTECTED MODE***

| **Registers at call:** | | **Return Registers:** |
|---|---|---|
| AH = 51h | | registers as returned by function |
| STACK: DWORD | selector:offset of function | |

**Details:** This call executes the specified function in protected mode with privilege level 0 (highest privilege).

**Conflicts:** Expansion Unit (Chapter 4)

**See also:** Functions 10h and 52h

---

**INT 15h Function 52h** *VMiX v2.?+*

***"sys_vm_init" - INITIALIZE PROTECTED-MODE ENVIRONMENT***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 52h | none |

**See also:** Functions 50h and 51h

## CHAPTER 47

# *DR-DOS TaskMAX*

TaskMAX is the task switcher included with DR-DOS version 6.0.

### INT 2Fh Function 2700h *DR-DOS 6.0 TaskMAX*
### *INSTALLATION CHECK*

| Registers at call: | Return Registers: |
|---|---|
| AX = 2700h | AL = status<br>00h not installed<br>FFh installed |

### INT 2Fh Function 2701h *DR-DOS 6.0 TaskMAX*
### *GET STATUS*

| Registers at call: | Return Registers: |
|---|---|
| AX = 2701h | AX = maximum simultaneous tasks<br>BX = index into TASK_IDS of current foreground task<br>CX = currently-active tasks<br>DX = version number (DR-DOS 6.0 = 0001h)<br>ES:SI -> TASK_IDS<br>ES:DI -> name table (array of 8-byte names, NUL-terminated if <8 chars) |

**Details:** Do not attempt to create a new task if CX == AX.

The task's index is its position on the task menu, while its ID is the position within the internal task name table.

**See also:** Functions 2714h and 2716h

### INT 2Fh Function 2702h *DR-DOS 6.0 TaskMAX*
### *GET PER-TASK EMS LIMIT*

| Registers at call: | Return Registers: |
|---|---|
| nnnAX = 2702h | DX = maximum pages INT 67h Function 42h will report available |

**Details:** TaskMAX does not limit EMS allocations other than by limiting the amount which is reported as being available at a given time.

**See also:** Function 2703h, EMS INT 67h Function 42h (Chapter 38)

### INT 2Fh Function 2703h *DR-DOS 6.0 TaskMAX*
### *SET PER-TASK EMS LIMIT*

| Registers at call: | Return Registers: |
|---|---|
| AX = 2703h<br>DX = maximum pages INT 67h Function 42h should report available | DX = new maximum for reporting |

**See also:** Function 2702h, EMS INT 67h Function 42h (Chapter 38)

---

### INT 2Fh Function 2704h
*DR-DOS 6.0 TaskMAX*
### *REGISTER/UNREGISTER TASK MANAGER*

**Registers at call:**
AX = 2704h
DL = subfunction
    00h unregister task manager
    01h register task manager

**Return Registers:**
DL = status
    00h registered
    01h unregistered

**Details:** A task manager replaces TaskMAX's menu system with its own user interface; while one is registered, the TaskMAX hotkeys and Ctrl-Alt-Del invoke the manager rather than the built-in menu system.

Unregister the task manager before terminating it.

**See also:** Function 2705h

---

### INT 2Fh Function 2705h
*DR-DOS 6.0 TaskMAX*
### *ENABLE/DISABLE DIRECT SWITCHING*

**Purpose:** Specify whether the keystrokes for switching to the next, the previous, or a specified task should be available to the user.

**Registers at call:**
AX = 2705h
DL = subfunction
    00h disable
    01h enable

**Return Registers:**
none

**Details:** This function should only be called by a registered task manager (see Function 2704h).

**See also:** Functions 2704h and 2706h

---

### INT 2Fh Function 2706h
*DR-DOS 6.0 TaskMAX*
### *SWITCH TO SPECIFIED TASK*

**Registers at call:**
AX = 2706h
DX = task index (see Function 2701h) of task to be activated

**Return Registers:**
DX = task index of previously-active task

**See also:** Functions 2705h, 2707h, and 2715h

---

### INT 2Fh Function 2707h
*DR-DOS 6.0 TaskMAX*
### *CREATE NEW TASK*

**Registers at call:**
AX = 2707h
DS:DX -> ASCIZ pathname of executable
ES:BX -> parameter block (see Table 47-1)
CX = number of ticks before automatic return to task manager (0000h = run until termination or explicitly switched)

**Return Registers:**
DX = new task's task index (FFFFh if task terminated)

**See also:** Functions 2706h and 2708h

**47-1 Format of parameter block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | reserved, should be 0000h |
| 02h | DWORD | pointer to command tail to be copied into child's PSP |
| 06h | DWORD | pointer to first FCB to be copied into child's PSP |
| 0Ah | DWORD | pointer to second FCB to be copied into child's PSP |

---

### INT 2Fh Function 2708h

*DR-DOS 6.0 TaskMAX*

### *DELETE TASK*

**Registers at call:**
AX = 2708h
DX = task index

**Return Registers:**
DX = FFFFh (task deleted)

**Details:** This call should only be used for abnormal task termination, after first checking for open files with Function 270Ch; it should not be used with programs that allocate EMS or XMS memory.

This function switches to the specified task first before deleting it.

**See also:** Function 2707h

---

### INT 2Fh Function 2709h

*DR-DOS 6.0 TaskMAX*

### *NAME TASK*

**Registers at call:**
AX = 2709h
DX = task index
DS:SI -> 8-byte name
  (8 NULs = remove name)

**Return Registers:**
AL = task flags
    00h ID unused or task terminated
    01h ID in use, task name table entry valid
    81h ID in use, task name fixed
BX = task ID
ES:DI -> name in task name table
  (see Function 2701h)

**Details:** The task retains the given name until it terminates or the name is removed by specifying a name of 8 NULs.

**See also:** Functions 2701h and 2707h

---

### INT 2Fh Function 270Ah

*DR-DOS 6.0 TaskMAX*

### *CONVERT TASK INDEX TO TASK ID*

**Registers at call:**
AX = 270Ah
DX = task index

**Return Registers:**
DX = task ID (FFFFh if index invalid)

**Details:** Task IDs stay constant, while indices can change when other tasks are deleted.

**See also:** Functions 2701h and 270Bh

---

### INT 2Fh Function 270Bh

*DR-DOS 6.0 TaskMAX*

### *CONVERT TASK ID TO TASK INDEX*

**Registers at call:**
AX = 270Bh
DX = task ID

**Return Registers:**
DX = task index (FFFFh if task not active)

**See also:** Function 270Ah

---

### INT 2Fh Function 270Ch

*DR-DOS 6.0 TaskMAX*

### *CHECK OPEN FILES*

**Registers at call:**
AX = 270Ch
DX = task index

**Return Registers:**
AX = number of files currently open for specified task

**See also:** Function 2708h

---

### INT 2Fh Function 270Dh

*DR-DOS 6.0 TaskMAX*

### *CHECK IF TASK RUNNING PRIMARY COMMAND INTERPRETER*

**Registers at call:**
AX = 270Dh
DX = task index

**Return Registers:**
DX = status
    0000h if primary command interpreter (COMMAND.COM, etc.) running
    0001h if not in root shell for task

**Details:** TaskMAX will return 0001h if the specified task has spawned another command interpreter with Function 2707h.
**See also:** Functions 2707h and 270Ch

---

### INT 2Fh Function 270Eh
*DR-DOS 6.0 TaskMAX*
### *GET/SET TEXT PASTE LEAD-IN*

**Registers at call:**
AX = 270Eh
CX = length of string (max 15 keystrokes, 0000h to get current string)
DS:SI -> pasting lead-in string (character/scan-code pairs)

**Return Registers:**
ES:DI -> current lead-in string

**Details:** The specified sequence of keystrokes is sent to the application before every line of a text-mode spreadsheet paste.
**See also:** Functions 270Fh, 2710h, and 2713h

---

### INT 2Fh Function 270Fh
*DR-DOS 6.0 TaskMAX*
### *GET/SET NUMERIC PASTE LEAD-IN*

**Registers at call:**
AX = 270Fh
CX = length of string (max 15 keystrokes, 0000h to get current string)
DS:SI -> pasting lead-in string (character/scan-code pairs)

**Return Registers:**
ES:DI -> current lead-in string

**Details:** The specified sequence of keystrokes is sent to the application before every number in a numeric-mode spreadsheet paste.
**See also:** Functions 270Eh, 2710h, 2711h, and 2713h

---

### INT 2Fh Function 2710h
*DR-DOS 6.0 TaskMAX*
### *GET/SET PASTE LINE TERMINATOR STRING*

**Registers at call:**
AX = 2710h
CX = length of string (max 15 keystrokes, 0000h to get current string)
DS:SI -> pasting terminator string (character/scan-code pairs)

**Return Registers:**
ES:DI -> current terminator string

**Details:** The specified sequence of keystrokes is sent to the application after every line of a spreadsheet paste operation.
**See also:** Functions 270Eh, 270Fh, and 2713h

---

### INT 2Fh Function 2711h
*DR-DOS 6.0 TaskMAX*
### *GET/SET NUMERIC PASTE DECIMAL POINT*

**Registers at call:**
AX = 2711h
DX = ASCII code for separator (FFFFh to get current)

**Return Registers:**
DL = current separator character

**See also:** Function 270Fh

---

### INT 2Fh Function 2712h
*DR-DOS 6.0 TaskMAX*
### *INITIATE EXPORTING TASK DATA*

**Registers at call:**
AX = 2712h
DX = task index

**Return Registers:**
none

**See also:** Function 2713h

---

**INT 2Fh Function 2713h** *DR-DOS 6.0 TaskMAX*

### *INITIATE PASTE OPERATION*

**Registers at call:**
AX = 2713h
DX = task index
CX = paste mode
    0000h alphanumeric
    0001h numeric
    0002h text

**Return Registers:**
none

**See also:** Functions 270Eh, 270Fh, 2710h, and 2711h

---

**INT 2Fh Function 2714h** *DR-DOS 6.0 TaskMAX*

### *GET SWAP SPACE INFO*

**Registers at call:**
AX = 2714h

**Return Registers:**
CX = total KB of swap space
DX = available KB of swap space

**See also:** Function 2701h

---

**INT 2Fh Function 2715h** *DR-DOS 6.0 TaskMAX*

### *SWITCH TO TASK MANAGER*

**Registers at call:**
AX = 2715h

**Return Registers:**
only after calling task is again selected

**See also:** Function 2706h

---

**INT 2Fh Function 2716h** *DR-DOS 6.0 TaskMAX*

### *GET PASTE BUFFER STATUS*

**Registers at call:**
AX = 2716h

**Return Registers:**
AX = 0000h if Functions 2716h-2718h supported
    CX = bytes in paste buffer
    DX = current generation number (updated after every copy operation)

**See also:** Functions 2701h, 2713h, 2714h, 2717h, and 2718h

---

**INT 2Fh Function 2717h** *DR-DOS 6.0 TaskMAX*

### *PASTE DATA DIRECTLY TO APPLICATION BUFFER*

**Registers at call:**
AX = 2717h
CX = bytes in destination buffer
ES:DI -> destination buffer

**Return Registers:**
AX = 0000h if function supported
    CX = bytes actually copied
     (FFFFh if buffer too small)
    DX = current generation number for paste buffer

**Details:** The destination buffer may be too small if another task adds more data to the paste buffer after the Function 2716h call but before this call.

**See also:** Functions 2713h, 2716h, and 2718h

---

**INT 2Fh Function 2718h** *DR-DOS 6.0 TaskMAX*

### *COPY DATA DIRECTLY INTO PASTE BUFFER*

**Registers at call:**
AX = 2718h
CX = bytes in source buffer
DS:SI -> source buffer (plain ASCII, lines terminated with CR LF)

**Return Registers:**
AX = 0000h if function supported
    CX = bytes actually copied
    DX = current generation number for paste buffer

**See also:** Functions 2712h, 2716h, and 2717h

## CHAPTER 48

# *DR Multiuser DOS*

Multiuser DOS (DRMDOS), from Digital Research Inc, is an MS-DOS compatible multitasking operating system designed to fill the gap left by the lack of multitasking capabilities of MS-DOS. DRMDOS provides a full range of services including file operations and time and date functions.

All of the INT E0h calls presented in this chapter are potentially in conflict with APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54), and Virus (Chapter 59). Since DRMDOS is a descendant of Concurrent CP/M, which itself is a descendant of CP/M-86, many of the INT E0h calls are also available under those earlier systems.

---

### INT E0h *CP/M-86, Concurrent CP/M, DR Multiuser DOS*

### *FUNCTION CALLS*

**Registers at call:**
CL = function number
DS,DX = parameters

**Return Registers:**
as appropriate for function

**Details:** Several functions are covered in more detail in following entries.

Most of these calls are also supported by Digital Research's DOS Plus v2.1; the unsupported functions are 26h,29h-2Bh,3Ah,3Dh-62h,71h-8Ch, 90h-92h,94h-97h,9Bh-ABh, and AEh-FFh.

**See also:** INT 21h Function 4459h, DOS Plus INT 21h Function E0h (Chapter 26)

**48-1 Values for function number:**

| | |
|---|---|
| 00h | terminate calling process (see also INT 21h Function 00h) |
| 01h | read a character |
| 02h | write character to default console (see also INT 21h Function 02h) |
| 03h | read character from default AUX (see also INT 21h Function 03h) |
| 04h | write character to default AUX (see also INT 21h Function 04h) |
| 05h | write character to default list device (see also INT 21h Function 05h) |
| 06h | perform raw I/O on default console (see also INT 21h Function 06h) |
| 07h | return default AUX input status |
| 08h | return default AUX output status |
| 09h | write string to default console (see also INT 21h Function 09h) |
| 0Ah | read string from default console (see also INT 21h Function 0Ah) |
| 0Bh | return default console input status (see also INT 21h Function 0Bh) |
| 0Ch | get BDOS release ID |
| 0Dh | reset all disk drives (see also INT 21h Function 0Dh) |
| 0Eh | set default drive (see also INT 21h Function 0Eh) |
| 0Fh | open file via FCB (see also INT 21h Function 0Fh) |
| 10h | close file via FCB (see also INT 21h Function 10h) |
| 11h | search for first matching file with FCB (see also INT 21h Function 11h) |

| | |
|---|---|
| 12h | search for next matching file with FCB (see also INT 21h Function 12h) |
| 13h | delete file via FCB (see also INT 21h Function 13h) |
| 14h | sequential read via FCB (see also INT 21h Function 14h) |
| 15h | sequential write via FCB (see also INT 21h Function 15h) |
| 16h | create file via FCB (see also INT 21h Function 16h) |
| 17h | rename file via FCB (see also INT 21h Function 17h) |
| 18h | get bit map of logged drives |
| 19h | get default drive (see also INT 21h Function 19h) |
| 1Ah | set DMA address offset |
| 1Bh | get default disk allocation vector (see also INT 21h Function 1Bh) |
| 1Ch | set default drive to read-only |
| 1Dh | get bit map of read-only drives |
| 1Eh | set file attributes via FCB (see also INT 21h Function 4301h) |
| 1Fh | get address of disk parameter block (see also INT 21h Function 1Fh) |
| 20h | get/set default user number |
| 21h | read random record via FCB (see also INT 21h Function 21h) |
| 22h | write random record via FCB (see also INT 21h Function 22h) |
| 23h | compute file size with FCB (see also INT 21h Function 23h) |
| 24h | get FCB random record number (see also INT 21h Function 24h) |
| 25h | reset specified drives |
| 26h | access specified drives (not in DR DOS Plus v2.1) |
| 27h | free specified drives |
| 28h | write random with FCB, zero fill (see also also INT 21h Function 28h) |
| 2Ah | lock records in FCB file (see also INT 21h Function 5Ch) |
| 2Bh | unlock records in FCB file (see also INT 21h Function 5Ch) |
| 2Ch | set BDOS multisector count |
| 2Dh | set BDOS error mode |
| 2Eh | get free space on disk |
| 2Fh | load, initialize, and jump to process (see also INT 21h Function 4Bh) |
| 30h | flush write-deferred buffers |
| 31h | get/set system variable (DOS Plus v2.1) |
| 32h | call BIOS (XIOS) character routine (see below) |
| 33h | set DMA address segment |
| 34h | get DMA buffer address |
| 35h | CP/M-86 allocate maximum memory |
| 36h | allocate maximum memory at specified segment |
| 37h | CP/M-86 allocate memory segment (see also INT 21h Function 48h) |
| 38h | allocate memory at specified segment |
| 39h | CP/M-86 free specified memory segment (see also INT 21h Function 49h) |
| 3Ah | CP/M-86 free all memory (not in DOS Plus v2.1) |
| 3Bh | load .CMD file into memory |
| 3Ch | (DOS Plus v2.1) call RSX program |
| 40h | (DR-NET) log on a server |
| 41h | (DR-NET) log off a server |
| 42h | (DR-NET) send a message |
| 43h | (DR-NET) receive a message |
| 44h | (DR-NET) get network status |
| 45h | (DR-NET) get requestor configuration table |
| 46h | (DR-NET) set compatibility attributes |

## INT E0h

| | |
|---|---|
| 47h | (DR-NET) get server configuration table |
| 48h | (DR-NET) set network error mode |
| 49h | (DR-NET) attach network |
| 4Ah | (DR-NET) detach network |
| 4Bh | (DR-NET) set default password |
| 4Ch | (DR-NET) get-set long timeout |
| 4Dh | (DR-NET) get parameter table |
| 50h | (DR-NET) get network information |
| 53h | get current time (see also INT 21h Function 2Ch) |
| 54h | set current time (see also INT 21h Function 2Dh) |
| 55h | get binary system date (see also INT 21h Function 2Ah) |
| 56h | set system date (see also INT 21h Function 2Bh) |
| 57h | allocate system flag |
| 58h | deallocate system flag |
| 59h | reserve memory in global area |
| 5Ah | lock physical drive |
| 5Bh | unlock physical drive |
| 5Ch | search path for executable file |
| 5Dh | load and execute command (see also INT 21h Function 4Bh) |
| 5Eh | get/set process exit code |
| 5Fh | set country information |
| 60h | get country information |
| 63h | truncate FCB file (see also INT 21h Function 28h) |
| 64h | create/update directory label |
| 65h | get directory label |
| 66h | get FCB date stamp and password mode |
| 67h | write extended FCB |
| 68h | set system date and time |
| 69h | get system date and time in binary |
| 6Ah | establish password for file access |
| 6Bh | get OS serial number |
| 6Ch | (DOS Plus v2.1) get/set program return code |
| 6Dh | get/set console mode |
| 6Eh | get/set string delimiter |
| 6Fh | write block to default console |
| 70h | write block to default list device |
| 71h | execute DOS-compatible function |
| 74h | set FCB time and date stamps |
| 80h | allocate memory |
| 82h | deallocate memory |
| 83h | poll I/O device |
| 84h | wait on system flag |
| 85h | set system flag |
| 86h | create message queue |
| 87h | open message queue |
| 88h | delete message queue |
| 89h | read from message queue |
| 8Ah | conditionally read from message queue |
| 8Bh | write to message queue |

| | |
|---|---|
| 8Ch | conditionally write to message queue |
| 8Dh | delay calling process |
| 8Eh | call process dispatcher (yield CPU) |
| 8Fh | terminate calling process (same as function 00h) |
| 90h | create a process |
| 91h | set calling process' priority |
| 92h | attach to default console |
| 93h | detach from default console |
| 95h | assign default console to process |
| 96h | interpret and execute commandline |
| 97h | resident procedure library |
| 98h | parse ASCII string into FCB (see also INT 21h Function 29h) |
| 99h | return default console |
| 9Ah | get address of system data (SYSDAT) |
| 9Bh | get system time and date |
| 9Ch | return calling process' descriptor |
| 9Dh | terminate process by name or PD address |
| 9Eh | attach to default list device |
| 9Fh | detach from default list device |
| A0h | select default list device |
| A1h | conditionally attach to default list device |
| A2h | conditionally attach to default console |
| A3h | get OS version number |
| A4h | get default list device |
| A5h | attach to default AUX |
| A6h | detach from default AUX |
| A7h | conditionally attach to default AUX |
| A8h | set default AUX |
| A9h | return default AUX |
| ACh | read block from default AUX |
| ADh | (DOS Plus v2.1) write block to default AUX |
| B0h | configure default AUX |
| B1h | get/set device control parameters |
| B2h | send Break through default AUX |
| B3h | allocate physical memory |
| B4h | free physical memory |
| B5h | map physical memory |
| B6h | nondestructive message queue read |
| B7h | timed wait on system flag |
| B8h | get/set I/O port mapping |
| B9h | set list device timeout |
| BAh | set AUX timeout value |
| BBh | execute XIOS service |
| BDh | (DR Multiuser DOS) delay |
| FFh | return 80386 to native mode |

**48-2 Values for DOS Plus v2.1 XIOS functions:**

| | |
|---|---|
| 00h | terminate program |
| 01h | *unknown* |
| 02h | check for console input status |

| | |
|---|---|
| 03h | read character from console |
| 04h | write character to console |
| 05h | write character to list device |
| 06h | write character to auxiliary device |
| 07h | read cahracter from auxiliary device |
| 0Fh | get list device status |
| 10h-14h | reserved |
| 15h | device initialization |
| 16h | check console output status |
| 17h-7Fh | reserved |
| **---BBC Acorn---** | |
| 80h | get XIOS version |
| 81h | get Tube semaphore |
| 82h | release Tube semaphore |
| 83h | select text/graphics |
| 84h | update B&W graphics rectangle |
| 85h | update color graphics rectangle |
| 86h | get/release/update mouse |
| 87h | get system error info |
| 88h | entry in CLOCK called by WatchDog RSP |
| 89h | BBC OSBYTE function |
| 8Ah | BBC OSWORD function |

### 48-3 DR Multiuser DOS Error Return Codes:

| | |
|---|---|
| 00h | no error |
| 01h | system call not implmented |
| 02h | illegal system call number |
| 03h | cannot find memory |
| 04h | illegal flag number |
| 05h | flag overrun |
| 06h | flag underrun |
| 07h | no unused Queue Descriptors |
| 08h | no free queue buffer |
| 09h | cannot find queue |
| 0Ah | queue in use |
| 0Ch | no free Process Descriptors |
| 0Dh | no queue access |
| 0Eh | empty queue |
| 0Fh | full queue |
| 10h | CLI queue missing |
| 11h | no 8087 in system |
| 12h | no unused Memory Descriptors |
| 13h | illegal console number |
| 14h | no Process Descriptor match |
| 15h | no console match |
| 16h | no CLI process |
| 17h | illegal disk number |
| 18h | illegal filename |
| 19h | illegal filetype |
| 1Ah | character not ready |

| | |
|---|---|
| 1Bh | illegal Memory Descriptor |
| 1Ch | bad return from BDOS load |
| 1Dh | bad return from BDOS read |
| 1Eh | bad return from BDOS open |
| 1Fh | null command |
| 20h | not owner of resource |
| 21h | no CSEG in load file |
| 22h | process Descriptor exists on Thread Root |
| 23h | could not terminate process |
| 24h | cannot attach to process |
| 25h | illegal list device number |
| 26h | illegal password |
| 28h | external termination occurred |
| 29h | fixup error upon load |
| 2Ah | flag set ignored |
| 2Bh | illegal auxilliary device number |

---

## INT E0h Function 03h

*DR Multiuser DOS*

### *"A_READ" - READ CHARACTER FROM AUX DEVICE*

| Registers at call: | Return Registers: |
|---|---|
| CL = 03h | AL = ASCII character |

**Details:** A_READ reads the next 8-bit character from the logical auxilliary input device (AUXn:); control is not returned to the calling process until a character has been read.

If another process owns AUX, this call blocks until the device becomes available.

**See also:** Functions 04h, 07h, A5h, and ACh; MS-DOS INT 21h Function 03h (Chapter 24)

---

## INT E0h Function 04h

*DR Multiuser DOS*

### *"A_WRITE" - WRITE CHARACTER TO AUX DEVICE*

| Registers at call: | Return Registers: |
|---|---|
| CL = 04h<br>DL = ASCII character | none |

**Details:** If another process owns AUX, this call blocks until the device becomes available.

**See also:** Functions 03h, 08h, A5h, and ADh; MS-DOS INT 21h Function 04h (Chapter 24)

---

## INT E0h Function 07h

*DR Multiuser DOS*

### *"A_STATIN" - GET INPUT STATUS OF AUX DEVICE*

**Purpose:** Determine whether the current AUX device has input available.

| Registers at call: | Return Registers: |
|---|---|
| CL = 07h | AL = status<br>00h not ready<br>FFh character available |

**See also:** Functions 03h and 08h

---

## INT E0h Function 08h

*DR Multiuser DOS*

### *"A_STATOUT" - GET OUTPUT STATUS OF AUX DEVICE*

**Purpose:** Determine whether the current AUX device is able to accept more output.

| Registers at call: | Return Registers: |
|---|---|
| CL = 08h | AL = status<br>00h not ready<br>FFh ready for output |

**See also:** Functions 04h and 07h

---

**INT E0h Function 59h** *Concurrent CP/M, DR Multiuser DOS*

***"S_MEMORY" - RESERVE MEMORY IN GLOBAL AREA***

**Registers at call:**
CL = 59h
DX = size in bytes

**Return Registers:**
AX = FFFFh if failed
AX = other if successful
ES:BX -> reserved memory

---

**INT E0h Function 86h** *Concurrent CP/M, DR Multiuser DOS*

***"Q_MAKE" - CREATE MESSAGE QUEUE***

**Registers at call:**
CL = 86h
DS:DX -> queue descriptor (see Table 48-5)

**Return Registers:**
AX = status (see Table 48-4)
CX = error code (see Table 48-3)

**See also:** Function 87h

**48-4 Values for DR Multiuser DOS function status:**

| | |
|---|---|
| 0000h | success |
| FFFFh | failure |

**48-5 Format of queue descriptor:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 WORDs | internal use, initialize to zeros |
| 04h | WORD | flags |
| 06h | 8 BYTEs | queue name |
| 0Eh | WORD | length of message |
| 10h | WORD | number of messages |
| 12h | 4 WORDs | internal use, initialize to zeros |
| 1Ah | WORD | offset in system area of buffer for messages |

---

**INT E0h Function 87h** *Concurrent CP/M, DR Multiuser DOS*

***"Q_OPEN" - OPEN MESSAGE QUEUE***

**Registers at call:**
CL = 87h
DS:DX -> queue parameter block (see Table 48-6)

**Return Registers:**
AX = status (see Table 48-4)
CX = error code (see Table 48-3)

**See also:** Functions 86h and 89h

**48-6 Format of queue parameter block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | internal use, initialize to zero |
| 02h | WORD | queue ID (set by Function 87h) |
| 04h | WORD | internal use, initialize to zero |
| 06h | WORD | offset of queue message buffer |
| 08h | 8 BYTEs | queue name |

---

**INT E0h Function 89h** *Concurrent CP/M, DR Multiuser DOS*

***"Q_READ" - READ MESSAGE QUEUE***

**Registers at call:**
CL = 89h
DS:DX -> queue parameter block (see Table 48-6)

**Return Registers:**
AX = status (see Table 48-4)
CX = error code (see Table 48-3)

**See also:** Functions 87h, 8Ah, and 8Bh

### INT E0h Function 8Ah
Concurrent CP/M,DR Multiuser DOS

### "Q_CREAD" - CONDITIONALLY READ MSG QUEUE

**Registers at call:**
CL = 8Ah
DS:DX -> queue parameter block (see Table 48-6)

**Return Registers:**
AX = status (see Table 48-4)
CX = error code (see Table 48-3)

**See also:** Functions 87h, 89h, and 8Ch

---

### INT E0h Function 8Bh
Concurrent CP/M,DR Multiuser DOS

### "Q_WRITE" - WRITE MESSAGE QUEUE

**Registers at call:**
CL = 8Bh
DS:DX -> queue parameter block (see Table 48-6)

**Return Registers:**
AX = status (see Table 48-4)
CX = error code (see Table 48-3)

**See also:** Functions 89h and 8Ch

---

### INT E0h Function 8Ch
Concurrent CP/M,DR Multiuser DOS

### "Q_CWRITE" - CONDITIONALLY WRITE MSG QUEUE

**Registers at call:**
CL = 8Ch
DS:DX -> queue parameter block (see Table 48-6)

**Return Registers:**
AX = status (see Table 48-4)
CX = error code (see Table 48-3)

**See also:** Functions 8Ah and 8Bh

---

### INT E0h Function 8Eh
Concurrent CP/M,DR Multiuser DOS

### "P_DISPATCH" - CALL DISPATCHER

**Purpose:** Allow other processes of the same or higher priority to run if they are ready.

**Registers at call:**
CL = 8Eh
DX = FFFFh (optional) to force dispatch

**Return Registers:**
none

**Details:** If DX=FFFFh, a dispatch is forced even if no other process is ready.

**See also:** Function 91h

---

### INT E0h Function 91h
Concurrent CP/M,DR Multiuser DOS

### "P_PRIORITY" - SET PROCESS PRIORITY

**Purpose:** Set the scheduling priority of the calling process; transient processes are initialized to priority C8h (200).

**Registers at call:**
CL = 91h
DL = new priority (00h highest to FFh lowest)

**Return Registers:**
none

**See also:** Function 8Eh

---

### INT E0h Function 93h
Concurrent CP/M,DR Multiuser DOS

### "C_DETACH" - DETACH FROM DEFAULT CONSOLE

**Registers at call:**
CL = 93h

**Return Registers:**
AX = status
  0000h successfully detached
  FFFFh detach failed

---

### INT E0h Function A5h
DR Multiuser DOS

### "A_ATTACH" - ATTACH AUX DEVICE

**Purpose:** This function attaches the default auxiliary device to the calling process unless it is already attached to another process, in which case the call blocks until the device becomes available.

**Registers at call:**
CL = A5h

**Return Registers:**
none

**Details:** This call should be used before attempting to read or write from the AUX device; however, the I/O calls internally call this function to ensure device ownership.
**Conflicts:** APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54), Virus (Chapter 59)
**See also:** Functions 03h, 04h, A6h, A7h, A8h, ACh, ADh, and B0h

---

### INT E0h Function A6h
*DR Multiuser DOS*

### *"A_DETACH" - DETACH FROM AUX DEVICE*

| Registers at call: | Return Registers: |
|---|---|
| CL = A6h | AX = status<br>0000h successfully detached<br>FFFFh detach failed<br>CX = error code (see Table 48-3) |

**See also:** Functions A5h and A7h

---

### INT E0h Function A7h
*DR Multiuser DOS*

### *"A_CATTACH" - CONDITIONALLY ATTACH TO AUX DEVICE*

**Purpose:** Attach the default auxiliary device to the calling process if it is available.

| Registers at call: | Return Registers: |
|---|---|
| CL = A7h | AX = status<br>0000h attached<br>FFFFh unable to attach |

**Details:** Unlike Function A5h, this function does not block if the device is already in use.
**See also:** Functions A5h, A6h, A8h, and B0h

---

### INT E0h Function A8h
*DR Multiuser DOS*

### *"A_SET" - SET DEFAULT AUX DEVICE NUMBER*

**Purpose:** Specify which physical device will become AUX.

| Registers at call: | Return Registers: |
|---|---|
| CL = A8h<br>DL = auxiliary device number | AX = status (see Table 48-4)<br>CX = error code (see Table 48-3) |

**See also:** Functions A5h and A9h

---

### INT E0h Function A9h
*DR Multiuser DOS*

### *"A_GET" - GET DEFAULT AUX DEVICE NUMBER*

**Purpose:** Determine which physical device is currently AUX.

| Registers at call: | Return Registers: |
|---|---|
| CL = A9h | AL = current default auxiliary device number |

**See also:** Function A8h

---

### INT E0h Function ACh
*DR Multiuser DOS*

### *"A_READBLK" - READ STRING FROM AUX DEVICE*

**Purpose:** Read characters from the default auxiliary (AUXn:) device into a buffer until the buffer is full or the device is no longer ready.

| Registers at call: | Return Registers: |
|---|---|
| CL = ACh<br>DS:DX -> character control block (see Table 48-7) | AX = number of characters read |

**Details:** If the device is initially not ready, this function blocks until at least one character has been read.

If another process owns AUX, this call blocks until the device becomes available.
**Conflicts:** APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54), Virus (Chapter 59)
**See also:** Functions 03h, A5h, and ADh

**48-7 Format of character control block (CHCB):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to character buffer |
| 04h | WORD | length of character buffer |

---

*DR Multiuser DOS*

### INT E0h Function ADh
### *"A_WRITEBLK" - WRITE STRING TO AUX DEVICE*

**Registers at call:**
CL = ADh
DS:DX -> character control block (see Table 48-7)

**Return Registers:**
AX = number of characters written

**Details:** This function does not return until at least one character has been written.

**See also:** Functions 04h, A5h, and ACh

---

*DR Multiuser DOS*

### INT E0h Function B0h
### *"A_CONFIG" - GET/SET AUX DEVICE PARAMETERS*

**Registers at call:**
CL = B0h
DX:DX -> AUX device parameter block (see below)

**Return Registers:**
AX = status
0000h successful
parameter block updated
FFFFh failed
CX = error code (see Table 48-3)

**See also:** Functions A5h and B1h

**48-8 Format of AUX device parameter block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | function (00h get, 01h set) |
| 01h | BYTE | baud rate (see below) FFh = don't change/unknown |
| 02h | BYTE | parity (see below) |
| 03h | BYTE | stop bits (00h one, 01h 1.5, 02h two, FFh unknown/don't change) |
| 04h | BYTE | data bits (05h-08h or FFh unknown/don't change) |
| 05h | BYTE | handshake (00h none, 01h DTS/DSR, 02h RTS/CTS, 04h XON/XOFF, FFh unknown/don't change) |
| 06h | BYTE | XON character, FFh unknown/don't change |
| 07h | BYTE | XOFF character, FFh unknown/don't change |

**48-9 Values for parity:**

| | |
|---|---|
| 00h | none |
| 01h | odd |
| 02h | none |
| 03h | even |
| 04h | stick parity bit |
| FFh | don't change/unknown |

**48-10 Values for baud rate:**

| | |
|---|---|
| 00h | 50 baud |
| 01h | 62.5 baud |
| 02h | 75 baud |
| 03h | 110 baud |
| 04h | 134.5 baud |
| 05h | 150 baud |
| 06h | 200 baud |
| 07h | 300 baud |
| 08h | 600 baud |

| | |
|---|---|
| 09h | 1200 baud |
| 0Ah | 1800 baud |
| 0Bh | 2000 baud |
| 0Ch | 2400 baud |
| 0Dh | 3600 baud |
| 0Eh | 4800 baud |
| 0Fh | 7200 baud |
| 10h | 9600 baud |
| 11h | 19200 baud |
| 12h | 38400 baud |
| 13h | 56000 baud |
| 14h | 76800 baud |
| 15h | 115200 baud |

---

## INT E0h Function B1h

*DR Multiuser DOS*

### *"A_CONTROL" - GET/SET AUX CONTROL PARAMETERS*

**Registers at call:**
CL = B1h
DS:DX -> AUX device control block
(see below)

**Return Registers:**
AX = status
0000h successful
control block updated
FFFFh failed
CX = error code (see Table 48-3)

**See also:** Functions B0h and B2h

**48-11 Format of AUX device control block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | function (00h get, 01h set) |
| 01h | BYTE | DTR state (00h low, 01h high, FFh unknown/don't change) |
| 02h | BYTE | RTS state (00h low, 01h high, FFh unknown/don't change) |
| 03h | BYTE | DSR state (00h low, 01h high, FFh unknown/don't change) |
| 04h | BYTE | CTS state (00h low, 01h high, FFh unknown/don't change) |
| 05h | BYTE | DCD state (00h low, 01h high, FFh unknown/don't change) |
| 06h | BYTE | RI state (00h inactive, 01h active, FFh unknown/don't change) |

---

## INT E0h Function B2h

*DR Multiuser DOS*

### *"A_BREAK" - SEND BREAK TO AUX DEVICE*

**Registers at call:**
CL = B2h
DX = duration of break in system ticks
(0001h-FFFFh)

**Return Registers:**
AX = status
0000h successful
break signal completed
FFFFh failed
CX = error code (see Table 48-3)

**Details:** If the AUX device is currently owned by another process, this call will block until the device becomes available.

**See also:** Functions A5h and B1h

---

## INT E0h Function BDh

*DR Multiuser DOS*

### *"P_DELAY" - DELAY EXECUTION*

**Registers at call:**
CL = BDh
DX = delay in system ticks

**Return Registers:**
after the delay elapses

**Details:** The length of a system tick is installation-dependent (typically 1/50 or 1/60 second); the length may be determined by reading the TICKSPERSEC value from the system data segment.

The actual delay before the process is rescheduled to run may be up to one tick longer than requested; the delay between rescheduling and actual execution cannot be predicted if higher-priority processes are awaiting a turn at the CPU.

**Conflicts:** APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54), Virus (Chapter 59)

**See also:** BIOS INT 15h Function 86h (Chapter 4), SND INT 1Ah Function FF01h (Chapter 22), FastGraph/Light INT 62h Function 0096h (Chapter 52), MS-DOS network support INT 2Fh Function 1224h (NI)

## INT E0h Function BDh

## CHAPTER 49

# *Other Multitaskers/Task Switchers*

This chapter discusses a number of other multitasking systems and task switchers, including Back&Forth, Cswitch, DOSSHELL, the DOS Task Switcher, DoubleDos, Headroon, OmniView, PC-MOS/386, CPtask, Software Carousel, SWELL, Switch-It, and several others.

## Back and Forth

Back & Forth is a task switcher by Progressive Solutions, Inc.

---

### INT 11h Function FFFEh, Subfunction FFFEh

*BACK&FORTH (before v1.62)*

***API***

| Registers at call: | Return Registers: |
|---|---|
| AX = FFFEh | |
| CX = FFFEh | |
| BX = function | |
| 00h installation check | AX = status<br>0001h BNFHIGH and BNFLOW both loaded<br>0003h only BNFHIGH loaded<br>else neither loaded |
| 01h *unknown* | DX:AX -> *unknown* |
| 02h *unknown* | |
| 03h *unknown* | |
| 04h *unknown* | |
| 05h *unknown* (switches current PSP segment and stack if BNFLOW has not yet announced itself installed) | AX = *unknown* |
| 06h *unknown* | |

**Conflicts:** CPU-generated INT 11h (Chapter 3), BIOS (Chapter 4), SDLP (Chapter 11), BNU FOSSIL (Chapter 12), Blank-It (Chapter 14), Borland C++ IDE (Chapter 54), RainboxFAX (Chapter 64)

**See also:** INT 12h Function FFFEh

---

### INT 12h Function FFFEh, Subfunction FFFEh

*Back&Forth v1.62+*

***API***

**Registers at call:**

AX = FFFEh
CX = FFFEh
BX = function
    00h installation check

02h build program ID list
ES:DI -> buffer of at least 100 bytes, to be filled with words

03h switch to specified task (task need not be open yet)
DX = two-letter program ID
04h *unknown*
05h *unknown*
06h get version (undoc)
07h *unknown*
08h get open tasks (undoc)
ES:DI -> task info buffer (see Table 49-1)
09h *unknown*

**Return Registers:**

AX = status
0001h installed
else not loaded

AX = number of programs defined
ES:DI buffer filled with AX words

AX = 0000h if task undefined

AX = version * 100 (v1.71 = 00ABh)

AX = number of open tasks (max 20)

**Conflicts:** BIOS (Chapter 4)
**See also:** INT 11h Function FFFEh

**49-1 Format of task information buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 21 BYTEs | ASCIZ task name |
| 15h | BYTE | hotkey shift state (as for INT 16h Function 02h) |
| 16h | WORD | hotkey scan code |
| 18h | WORD | program ID |

## Cswitch

Cswitch is a set of multitasking functions by Herb Rose. As with any program using interrupts in the range 60h through 67h, there are numerous potential conflicts, which are listed in Table 2-3.

---

### INT 62h Function 01h
*Cswitch*
### *GIVE UP REST OF TIME-SLICE*

**Registers at call:**
AH = 01h

**Return Registers:**
none

**See also:** Functions 05h and 06h, TopView INT 15h Function 1000h (Chapter 44), MS Windows INT 2Fh Function 1680h (Chapter 43)

---

### INT 62h Function 02h
*Cswitch*
### *WAIT FOR SEMAPHORE*

**Registers at call:**
AH = 02h
DX = semaphore number (0-63)

**Return Registers:**
AX = status
FFFFh bad semaphore number
else success

**See also:** Functions 03h and 04h

---

### INT 62h Function 03h
*Cswitch*
### *CHECK SEMAPHORE*

**Registers at call:**
AH = 03h
DX = semaphore number (0-63)

**Return Registers:**
AX = status
FFFFh not owned
else owned

**See also:** Functions 02h and 04h

INT 62h Function 01h

---

### INT 62h Function 04h
### *TRIGGER SEMAPHORE*

*Cswitch*

**Registers at call:**
AH = 04h
DX = semaphore number (0-63)

**Return Registers:**
AX = status
FFFFh bad semaphore number
else success

**See also:** Functions 02h and 03h

---

### INT 62h Function 05h
### *SLEEP*

*Cswitch*

**Registers at call:**
AH = 05h
BX = seconds to sleep

**Return Registers:**
after sleep interval expires

**See also:** Functions 01h, 06h, and 08h

---

### INT 62h Function 06h
### *SUSPEND*

*Cswitch*

**Registers at call:**
AH = 06h

**Return Registers:**
none

**See also:** Functions 05h and 08h

---

### INT 62h Function 07h
### *SPAWN*

*Cswitch*

**Registers at call:**
AH = 07h
ES:BX -> function address to start executing at
CX = priority (1-10)

**Return Registers:**
AX = status
FFFDh no free memory control blocks
FFFEh no free task control blocks
FFFFh not enough memory to create new task stack
>0 the tcb number of the new task, indicating no error

**See also:** Functions 0Fh and 10h

---

### INT 62h Function 08h
### *WAKE UP TASK*

*Cswitch*

**Registers at call:**
AH = 08h
BX = tcb identifier

**Return Registers:**
none

**See also:** Functions 05h and 06h

---

### INT 62h Function 09h
### *SET PRIORITY*

*Cswitch*

**Registers at call:**
AH = 09h
BX = new base priority (1-10)

**Return Registers:**
none

**Details:** The lower the priority is numerically, the more often the task will run.

---

### INT 62h Function 0Ah
### *TEST MESSAGE QUEUE*

*Cswitch*

**Registers at call:**
AH = 0Ah
DX = queue number (0-63)

**Return Registers:**
AX = status
FFFFh bad queue number
0000h nothing on queue
else bytes in first queued message

**See also:** Functions 0Bh and 0Ch

### INT 62h Function 0Bh

*Cswitch*

#### *SEND MESSAGE*

**Registers at call:**
AH = 0Bh
CX = number of bytes to write
DS:SI -> buffer
DX = queue number (0-63)

**Return Registers:**
AX = status
FFFEh triggered by arriving data, repeat call
FFFFh bad queue number
0000h no message was on queue
else number of bytes in message

**See also:** Functions 0Ah and 0Ch

### INT 62h Function 0Ch

*Cswitch*

#### *READ MESSAGE*

**Registers at call:**
AH = 0Ch
CX = number of bytes to read
DS:SI -> buffer
DX = queue number (0-63)

**Return Registers:**
AX = status
FFFFh bad queue number
else number of bytes transferred

**See also:** Functions 0Ah and 0Bh

### INT 62h Function 0Dh

*Cswitch*

#### *DON'T ALLOW TASK TO BE SWAPPED OUT*

**Registers at call:**
AH = 0Dh

**Return Registers:**
none

**See also:** Function 0Eh

### INT 62h Function 0Eh

*Cswitch*

#### *ALLOW TASK TO BE SWAPPED OUT*

**Registers at call:**
AH = 0Eh

**Return Registers:**
none

**Conflicts:** see Table 2-3
**See also:** Function 0Dh

### INT 62h Function 0Fh

*Cswitch*

#### *LOAD AND RUN PROGRAM FROM DISK*

**Registers at call:**
AH = 0Fh
ES:BX -> command line
CX = priority (1-10)
DX = background flag (nonzero allows loading to EMS)

**Return Registers:**
AX = status
0000h task loader queue is full
0001h no error

**Conflicts:** see Table 2-3
**See also:** Functions 07h, 10h, and 13h

### INT 62h Function 10h

*Cswitch*

#### *TERMINATE SPAWNED PROGRAM*

**Registers at call:**
AH = 10h

**Return Registers:**
none

**See also:** Functions 07h and 0Fh

### INT 62h Function 11h

*Cswitch*

#### *GET TCB INFORMATION*

**Registers at call:**
AH = 11h
ES:BX -> a pointer which will be set to the tcb address

**Return Registers:**
AX = tcb indentifier

**See also:** Function 12h

---

### INT 62h Function 12h

*Cswitch*

***GET TCB ADDRESS***

**Registers at call:**
AH = 12h
ES:BX -> a pointer which will be set to the tcb table address

**Return Registers:**
AX = tcb indentifier

**See also:** Function 11h

---

### INT 62h Function 13h

*Cswitch*

***CHECK STATUS OF PREVIOUS LOAD_TASK***

**Registers at call:**
AH = 13h

**Return Registers:**
AX = status
FFFCh no Memory Control Blocks available
FFFDh no TCBs available
FFFEh insufficient memory
FFFFh cannot open file
0000h load in progress (not done yet)
else tcb indentifier

**Conflicts:** see Table 2-3
**See also:** Function 0Fh

## DOSSHELL

DOSSHELL is a menu utility that has been supplied with MS-DOS and IBM versions since 4.0. The implementation in version 4 differed substantially from that in later versions. SHELLB.COM and SHELLC.EXE are parts of the DOS 4.x shell. The DOS 5 and DOS 6 DOSSHELLs consist of the main program DOSSHELL and the optional task switcher code in DOSWAP.EXE.

---

### INT 2Fh Function 1900h

*DOS 4.x only SHELLB.COM (undocumented)*

***INSTALLATION CHECK***

**Registers at call:**
AX = 1900h

**Return Registers:**
AL = status
00h not installed
FFh installed

---

### INT 2Fh Function 1901h

*DOS 4.x only SHELLB.COM (undocumented)*

***SHELLC.EXE INTERFACE***

**Registers at call:**
AX = 1901h
BL = SHELLC type
00h transient
01h resident
DS:DX -> far call entry point for resident SHELLC.EXE

**Return Registers:**
ES:DI -> SHELLC.EXE workspace within SHELLB.COM

---

### INT 2Fh Function 1902h

*DOS 4.x only SHELLB.COM (undocumented)*

***COMMAND.COM INTERFACE***

**Registers at call:**
AX = 1902h
ES:DI -> ASCIZ full filename of current batch file, with at least the final filename element uppercased
DS:DX -> buffer for results

**Return Registers:**
AL = status
00h failed (see notes below)
FFh success, then:
memory at DS:[DX+1] onwards filled as:

| | | |
|---|---|---|
| DX+1: | BYTE | count of bytes of PSC |
| DX+2: | N BYTEs | Program Start Command text |
| DX+X: | BYTE | 0Dh terminator |

**Details:** COMMAND.COM executes the result of this call in preference to reading a command from a batch file. Thus the batch file does not advance in execution for so long as SHELLB provides PSCs from its workspace. The PSCs are planted in SHELLB workspace by SHELLC, the user menu interface. The final PSC of a sequence is finished with a GOTO COMMON, which causes a loop back in the batch file which called SHELLC so as to execute SHELLC again. The check on batch file name permits PSCs to CALL nested batch files while PSCs are still stacked up for subsequent execution.

This call may fail because either (a) the final filename element quoted at ES:DI does not match the identity of the shell batch file quoted as a parameter of the most recent call of the SHELLB command, or (b) no more Program Start Commands are available.

---

## INT 2Fh Function 1903h
### *COMMAND.COM interface*

*DOS 4.x only SHELLB.COM*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1903h | AL = status |
| ES:DI -> ASCIZ batch file name as for Function 1902h | FFh if quoted batch file name matches last SHELLB parameter<br>00h if it does not |

---

## INT 2Fh Function 1904h
### *SHELLB transient to TSR intrface*

*DOS 4.x only SHELLB.COM*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 1904h | ES:DI -> name of current shell batch file (see Table 49-2) |

**49-2 Format of batch file name:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of bytes of name following |
| 02h | N BYTEs | (8 max) uppercase name of shell batch file |

---

## INT 2Fh Function 4A05h
### *DOSSWAP* - *TASK SWITCHING API*

*DOS 5+ DOSSHELL*
*(undocumented)*

**Registers at call:**
AX = 4A05h
SI = subfunction
  0000h *reset*
  0001h *unknown*
    ES:BP -> 80-byte buffer containing *unknown*
  0002h *unknown*
  0003h *unknown*
  0004h *unknown*
    BL = *unknown*
  0005h *unknown*

**Return Registers:**
none or *unknown*

**Registers at call:**
AX = 4A05h
SI = subfunction
  0006h get *unknown*

  0007h get *unknown*

  0008h get *unknown*

  0009h get *unknown*

  000Ah *unknown*

**Return Registers:**

BL = length of buffer
ES:BP -> buffer
containing unknown
ES:SI -> *unknown*
000Bh get *unknown*
AX = *unknown*
000Ch *unknown*
BL = *unknown*
DX:AX -> *unknown* (internal control data of some kind)

ES:SI -> *unknown* (apparently identical to function 0006h)

AX = *unknown*

if BL nonzero on entry
  DX:AX -> *unknown*
if BL = 00h on entry
  ES:SI -> *unknown*

**Details:** DOSSHELL chains to the previous handler if SI is not one of the values listed above.

The DOSSWAP.EXE module calls functions 03h,04h,05h,07h,08h,09h,0Ch.

The Windows 3.1 DSWAP.EXE and WSWAP.EXE task switchers use these calls.

**See also:** Function 4B01h

## DOS Task Switcher

The DOS Task Switcher is a rudimentary multiprocessing capability that made its appearance in MS-DOS at version 5.0. It is an integral part of DOSSHELL, but can also be used by other applications that meet its rather arcane API requirements.

---

### INT 2Fh Function 4B01h

*DOS 5+ TASK SWITCHER*

***BUILD CALLOUT CHAIN***

*Callout*

**Registers at call:**
AX = 4B01h
CX:DX -> task switcher entry point (see Table 49-5)
ES:BX = 0000h:0000h

**Return Registers:**
ES:BX -> callback info structure (see below) or 0000h:0000h

**Details:** This function is called by the task switcher, and hooked by clients which require notification of task switcher activities. The call must first be passed on to the prior handler with registers unchanged using a simulated interrupt. On return, the client must build a callback info structure and store the returned ES:BX in the “next” field, then return the address of its own callback info structure.

A client program must add itself to the notification chain if it provides services to other programs; before terminating, it must remove itself from the chain by calling the task switcher's entry point with AX=0005h (see Function 4B02h).

The task switcher entry point should not be saved, as it is subject to change and will be provided on any notification call.

The Windows 3.1 Standard Mode supports this API.

**See also:** Function 4B02h, MS Windows Function 160Bh (Chapter 43)

**49-3 Format of callback info structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to next callback info structure |
| 04h | DWORD | pointer to notification function (see below) |

| | | |
|---|---|---|
| 08h | DWORD | reserved |
| 0Ch | DWORD | address of zero-terminated list of API info structures (see Function 4B02h) |

**49-4 Notification function calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AX = function | |
| 0000h switcher initialization | AX = status<br>0000h if OK to load<br>nonzero to abort task switcher |
| 0001h query suspend<br>BX = session ID | AX = status<br>0000h if OK to switch session<br>0001h if not |
| 0002h suspend session<br>BX = session ID<br>interrupts disabled | AX = status<br>0000h if OK to switch session<br>0001h if not |

| Registers at call: | Return Registers: |
|---|---|
| AX = function | |
| 0003h activate session<br>BX = session ID<br>CX = session status flags<br>bit 0: set if first activation of session<br>bits 1-15: reserved (0)<br>interrupts disabled | AX = 0000h |
| 0004h session active<br>BX = session ID<br>CX = session status flags (see above) | AX = 0000h |
| 0005h create session<br>BX = session ID | AX = status<br>0000h if OK to create session<br>0001h if not |

| Registers at call: | Return Registers: |
|---|---|
| AX = function | |
| 0006h destroy session<br>BX = session ID | AX = 0000h |
| 0007h switcher termination<br>BX = flags<br>bit 0: set if calling switcher is only switcher loaded<br>bits 1-15: reserved (0) | AX = 0000h<br>ES:DI -> task switcher entry point (see Table 49-5) |

**Details:** Function 0000h is generally called by the program which controls or invokes the task switcher, rather than by the task switcher itself; the entry point supplied to this function is not necessarily the entry point to the task switcher itself, and may be 0000h:0000h. If any client indicates that loading is not possible, all clients will be called with function 0007h; thus it is possible for a client to receive a termination notice without a corresponding initialization notice.

Except for functions 0002h and 0003h, the notification handler is called with interrupts enabled and may make any INT 21h function call; interrupts must not be enabled in functions 0002h and 0003h.

Function 0007h may be called with ES:DI = 0000h:0000h if the entry point is no longer valid.

## INT 2Fh Function 4B01h

## INT 2Fh Function 4B02h

*DOS 5+ TASK SWITCHER*

### *INSTALLATION CHECK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4B02h | ES:DI = 0000h:0000h if task switcher not loaded |
| BX = 0000h | ES:DI -> task switcher entry point (see Table 49-5) if loaded |
| ES:DI = 0000h:0000h | AX = 0000h |

**Details:** The returned entry point is that for the most-recently loaded task switcher; the entry points for prior task switchers may be determined with the "get version" call (see Table 49-5).

This function is supported by PC Tools v8+ CPTASK.

**See also:** Function 4B03h, DOSSWAP Function 4A05h (above)

**49-5 task switcher entry point calling convention:**

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = function | |
| 0000h get version | CF clear if successful<br>AX = 0000h<br>ES:BX -> task switcher version structure (see Table 49-6)<br>CF set if unsupported function |
| 0001h test memory region<br>ES:DI -> first byte to be tested<br>CX = size of region to test | CF clear if successful<br>AX = memory type of tested region<br>0000h global<br>0001h global and local<br>0002h local (replaced on session switch)<br>CF set if unsupported function |
| 0002h suspend switcher<br>ES:DI -> new task switcher's entry point | CF clear if successful<br>AX = state<br>0000h switcher has been suspended<br>0001h switcher not suspended, new switcher must abort<br>0002h switcher not suspended, but new switcher may run anyway<br>CF set if unsupported function |

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = function | CF clear if successful<br>AX = 0000h<br>CF set if unsupported function |
| 0003h resume switcher<br>ES:DI -> new task switcher's entry point | |
| 0004h hook notification chain<br>ES:DI -> callback info structure to be added to chain (see Table 49-3) | |
| 0005h unhook notification chain<br>ES:DI -> callback info structure to be removed from chain (see Table 49-3) | |

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = function | CF clear if successful |
| 0006h query API support<br>BX = asynchronous API identifier | AX = 0000h<br>ES:BX -> API info structure (see Table 49-7) for the client which provides the highest level of support CF set if unsupported function |

### 49-6 Format of task switcher version structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | major version of supported protocol (current protocol is 1.0) |
| 02h | WORD | minor version of supported protocol |
| 04h | WORD | major version of task switcher |
| 06h | WORD | minor version of task switcher |
| 08h | WORD | task switcher ID (see Function 4B03h) |
| 0Ah | WORD | operation flags<br>bit 0: set if task switcher disabled<br>bits 1-15: reserved (0) |
| 0Ch | DWORD | pointer to ASCIZ task switcher name<br>("MS-DOS Shell Task Switcher" for DOSSHELL task switcher) |
| 10h | DWORD | pointer to previous task switcher's entry point or 0000h:0000h |

### 49-7 Format of API info structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | size of structure in bytes (000Ah) |
| 02h | WORD | API identifier<br>0001h NetBIOS<br>0002h 802.2<br>0003h TCP/IP<br>0004h LAN Manager named pipes<br>0005h Novell NetWare IPX |
| 04h | WORD | major version \ of highest version of API for which the support |
| 06h | WORD | minor version / level specified in the next field is provided |
| 08h | WORD | support level<br>0001h minimal support<br>0002h API-level support<br>0003h switcher compatibility<br>0004h seamless compatibility |

---

## INT 2Fh Function 4B03h

*DOS 5+ TASK SWITCHER*

### *ALLOCATE SWITCHER ID*

**Registers at call:**
AX = 4B03h
ES:DI -> task switcher entry point (see Table 49-5)

**Return Registers:**
AX = 0000h
BX = switcher ID (0001h-000Fh), or 0000h if no more available

**Details:** If a task switcher has determined that it is the first to be loaded, it must allocate an identifier for itself and provide this function to all subsequent task switchers; if it is not the first to be loaded, it must call this function to allocate an ID. The switcher ID is used as the high four bits of all session identifiers to ensure unique session IDs.

If no more switcher IDs are available, the new task switcher making the call must terminate or disable itself.

The task switcher providing the identifiers may call the new task switcher's entry point as needed.

This call is available from within DOSSHELL even if the task switcher is not installed. It is also supported by PC Tools v8+ CPTASK, but apparently always returns an ID of 0000h.

**See also:** Functions 4B02h and 4B04h

---

## INT 2Fh Function 4B04h

*DOS 5+ TASK SWITCHER*

### *FREE SWITCHER ID*

**Registers at call:**
AX = 4B04h

BX = switcher ID
ES:DI -> task switcher entry point
(see Table 49-5)

**Return Registers:**
AX = 0000h
BX = status
0000h successful
other error (invalid ID or ID not allocated)

**Details:** This function is called by a task switcher when it exits, unless it was the first loaded and is providing the support for Functions 4B03h and 4B04h.

The task switcher providing the identifiers may call the terminating task switcher's entry point as needed.

This call is available from within DOSSHELL even if the task switcher is not installed. It is also supported by PC Tools v8+ CPTASK, but appears to return sucessfully no matter which ID is given.

**See also:** Functions 4B02h and 4B03h

---

### INT 2Fh Function 4B05h
### *IDENTIFY INSTANCE DATA*

*DOS 5+ TASK SWITCHER*
*Callout*

**Registers at call:**
AX = 4B05h
ES:BX = 0000h:0000h
CX:DX -> task switcher entry point (see Table 49-5)

**Return Registers:**
ES:BX -> startup info structure (see Table 49-8) or 0000h:0000h

**Details:** This function is called by the task switcher. Clients with instance data should hook this call and pass it through to the previous handler with unchanged registers using a simulated interrupt. On return, the client should create a startup info structure (see Table 49-8), store the returned ES:BX in the "next" field, and return the address of the created structure in ES:BX.

All MS-DOS function calls are available from within this call.

**See also:** Function 4B02h, MS Windows Functions 1605h and 160Bh (Chapter 43)

**49-8 Format of startup info structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | major, minor version of info structure (03h,00h) |
| 02h | DWORD | pointer to next startup info structure or 0000h:0000h |
| 06h | DWORD | 0000h:0000h (ignored) |
| 0Ah | DWORD | ignored |
| 0Eh | DWORD | pointer to instance data records (see Table 49-9) |

**49-9 Format of one instance data record in array:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | address of instance data (end of array if 0000h:0000h) |
| 04h | WORD | size of instance data |

## DoubleDOS

DoubleDOS is an early DOS multitasker by SoftLogic Solutions, Inc., which permits two programs to execute concurrently.

---

### INT 21h Function E0h
### *MENU CONTROL*

*DoubleDOS*

**Registers at call:**
AH = E0h
AL = subfunction
01h exchange tasks
73h resume invisible job if suspended
74h kill other job
75h suspend invisible job

**Return Registers:**
none

**Conflicts:** DOS Plus (Chapter 26), OS/286 (Chapter 34), Virus (Chapter 59), Novell NetWare (NI)

**See also:** Function F0h

### INT 21h Function E1h

*DoubleDOS*

### *CLEAR KEYBOARD BUFFER FOR CURRENT JOB*

**Registers at call:**
AH = E1h

**Return Registers:**
none

**Conflicts:** OS/286 (Chapter 34), Virus (Chapter 59), Novell NetWare (NI)
**See also:** Functions E2h, E3h, E8h, and F1h

### INT 21h Function E2h

*DoubleDOS*

### *SEND CHARACTER TO KEYBOARD BUFFER OF OTHER JOB*

**Registers at call:**
AH = E2h
AL = character

**Return Registers:**
AL = status
    00h successful
    01h buffer full (128 characters)

**Conflicts:** OS/286 (Chapter 34), Novell NetWare (NI)
**See also:** Functions E1h, E3h, E8h, and F2h

### INT 21h Function E3h

*DoubleDOS*

### *ADD CHARACTER TO KEYBOARD BUFFER OF CURRENT JOB*

**Registers at call:**
AH = E3h
AL = character

**Return Registers:**
AL = status
    00h successful
    01h buffer full (128 characters)

**Conflicts:** OS/286 (Chapter 34), Novell NetWare (NI)
**See also:** Functions E1h, E2h, E8h, and F3h

### INT 21h Function E400h

*DoubleDOS*

### *INSTALLATION CHECK/PROGRAM STATUS*

**Registers at call:**
AX = E400h

**Return Registers:**
AL = status
    00h if DoubleDOS not present
    01h if running in visible DoubleDOS partition
    02h if running in the invisible partition

**Conflicts:** OS/286 (Chapter 34), Virus (Chapter 59), Novell NetWare (NI)
**See also:** Functions E5h and F400h

### INT 21h Function E5h

*DoubleDOS*

### *OTHER PROGRAM STATUS*

**Registers at call:**
AH = E5h

**Return Registers:**
AL = status
    00h no program in other partition
    01h program in other partition is running
    02h program in other partition is suspended

**Conflicts:** OS/286 (Chapter 34), Novell NetWare (NI)
**See also:** Functions E400h and F5h

### INT 21h Function E8h

*DoubleDOS*

### *SET/RESET KEYBOARD CONTROL FLAGS*

**Registers at call:**
AH = E8h
AL = program for which to set flags
    00h this program, 01h other program
DX = keyboard control flags (see Table 49-10)

**Return Registers:**
DX = previous flags

**Details:** Disabling Ctrl-PrtSc will allow the program to intercept the keystroke; disabling any of the other keystrokes disables them completely.

This function is identical to Function F8h.

**Conflicts:** OS/286 (Chapter 34), Alloy NTNX (NI), Novell NetWare (NI)

**See also:** Functions E1h, E2h, E3h, and F8h

**49-10 Bitfields for keyboard control flags:**

| | |
|---|---|
| bit 0 | menu |
| bit 1 | exchange |
| bit 2 | entire keyboard enable/disable |
| bit 3 | Ctrl-C |
| bit 4 | Ctrl-PrtSc |
| bit 5 | Alt/Erase |
| bit 6 | Ctrl-Break |
| bit 7 | Ctrl-NumLock |
| bit 8 | shift-PrtSc |
| bit 9-13 | undefined |
| bit 14 | cancel key (clear keyboard buffer) |
| bit 15 | suspend key |

**Note:** Setting a bit enables the corresponding key or operation, clearing a bit disables it.

---

### INT 21h Function E9h
*DoubleDOS*

### *SET TIMESHARING PRIORITY*

| Registers at call: | Return Registers: |
|---|---|
| AH = E9h | AL = priority setting |
| AL = priority (see Table 49-11) | |

**Details:** This function is identical to Function F9h.

**Conflicts:** OS/286 (Chapter 34), Novell NetWare (NI)

**See also:** Functions EAh, EBh, and F9h

**49-11 Timesharing priorities:**

| | |
|---|---|
| 00h | visible program gets 70%, invisible gets 30% (default) |
| 01h | visible program gets 50%, invisible gets 50% |
| 02h | visible program gets 30%, invisible gets 70% |
| 03h | Top program gets 70%, bottom program gets 30% |
| 04h | Top program gets 30%, bottom program gets 70% |
| 05h | get current priority |

---

### INT 21h Function EAh
*DoubleDOS*

### *TURN OFF TASK SWITCHING*

| Registers at call: | Return Registers: |
|---|---|
| AH = EAh | task switching turned off |

**Conflicts:** OS/286 (Chapter 34), Novell NetWare (NI)

**See also:** Functions E9h, EBh, and FAh; INT FAh

---

### INT 21h Function EBh
*DoubleDOS*

### *TURN ON TASK SWITCHING*

| Registers at call: | Return Registers: |
|---|---|
| AH = EBh | task switching turned on |

**Conflicts:** OS/286 (Chapter 34), Novell NetWare (NI)

**See also:** Functions E9h, EAh, and FBh; INT FBh

### INT 21h Function ECh

*DoubleDOS*

### *GET VIRTUAL SCREEN ADDRESS*

**Purpose:** Determine the address of the virtual screen to which the program should write instead of the actual video memory, so that the multitasked programs do not interfere with each other's output.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = ECh | ES = segment of virtual screen |

**Details:** The screen address can change if task-switching is enabled!

This function is identical to Function FCh.

**Conflicts:** OS/286 (Chapter 34), Virus (Chapter 59), Novell NetWare (NI)

**See also:** Function FCh, INT FCh, TopView INT 10h Function FEh (Chapter 44)

### INT 21h Function EEh

*DoubleDOS*

### *GIVE AWAY TIME TO OTHER TASKS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = EEh<br>AL = number of 55ms time slices to give away | returns after giving away time slices |

**Conflicts:** Virus (Chapter 59), Novell NetWare (NI)

**See also:** Function FEh, INT FEh

### INT 21h Functions F0h to FEh

*DoubleDOS*

### *DUPLICATED CALLS*

These functions are identical to Functions E0h through EEh, so see those entries above.

**Conflicts:** CED (Chapter 13), DOSED (Chapter 13), PCMANAGE/DCOMPRES (Chapter 16), DOS v2.11 OEM Functions (Chapter 24), DOS/4GW (Chapter 35), GO32 (Chapter 37), VDEFEND (Chapter 59), FLU_SHOT+ (Chapter 59), Virus (Chapter 59), Novell NetWare (NI), TopWare Network (NI)

### INT 58h to INT 5Fh

*DoubleDOS*

### *RELOCATED IRQ0 THROUGH IRQ7*

**Purpose:** DoubleDOS reprograms the interrupt controller to use this alternate range of interrupts so that DoubleDOS can maintain control even when programs override the keyboard INT 09h.

**Conflicts:** GSS*CGI (Chapter 8), HP 95LX (Chapter 20), DESQview (Chapter 44), SitBack (Chapter 66), Cluster adapter (NI), NetBIOS (NI), Microsoft Network Transport Layer (NI), Alloy NTNX (NI), ISOLAN (NI), TOPS (NI), ATALK.SYS (NI), IBM 802.2 interface (NI), $25 LAN (NI)

**See also:** Hardware INT 08h to INT 0Fh (Chapter 3)

### INT F4h

*DoubleDOS*

### *GIVE UP REST OF CURRENT CLOCK TICK AND ALL OF NEXT TICK*

**See also:** INT 21h Function EEh, INT FEh

### INT F5h to INT F9h

*DoubleDOS*

*UNDETERMINED FUNCTIONS*

**Conflicts:** FSBBS (Chapter 66)

### INT FAh

*DoubleDOS*

### *TURN OFF TIMESHARING*

**See also:** INT 21h Function EAh, INT FBh

### INT FBh

*DoubleDOS*

### *TURN ON TIMESHARING*

**See also:** INT 21h Function EBh, INT FAh

### INT FCh

*DoubleDOS*

### *GET CURRENT SCREEN BUFFER ADDRESS*

**Purpose:** Determine the address of the virtual screen to which the program should write instead of the actual video memory, so that the multitasked programs do not interfere with each other's output.

**Return Registers:**
ES = segment of display buffer

**Details:** The display buffer may be moved if multitasking is enabled.
**See also:** INT 21h Function ECh, INT FBh

---

**INT FDh** *DoubleDOS*
*UNDETERMINED FUNCTION*
**Conflicts:** TFPCX (Chapter 12)

---

**INT FEh** *DoubleDOS*
***GIVE UP TIME***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AL = number of 55ms time slices to give away | after other program (if active) has run |

**Conflicts:** AT BIOS (Chapter 4), Turbo Debugger (Chapter 57)
**See also:** INT 21h Function EEh, INT F4h

## Helix Software Headroom

Headroom is a TSR manager/task switcher by Helix Software.

---

**INT 21h Function 4C57h** *Headroom*
*UNDETERMINED FUNCTION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4C57h | never returns |
| DS:DX -> target address | |

**Details:** This function jumps to the target address instead of terminating the program.
**Conflicts:** MS-DOS (Chapter 24)
**See also:** Function 5758h

---

**INT 21h Function 5758h** *Headroom (undocumented)*
***API***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5758h | |
| BL = function | |
| 00h *unknown* | |
| 01h get Headroom location | CF clear if installed<br>AX = PSP segment of Headroom TSR<br>BX = paragraphs of memory used by Headroom<br>CF set if not (normal DOS return) |
| 02h get INT 21h handler | |

**Details:** This function is also used as an installation check.
CF clear
ES:BX -> Headroom's INT 21h handler
**Details:** This function also sets an unknown flag.
**Registers at call:**
AX = 5758h
BL = function
03h *launch application*
DS:SI -> 233-byte application record.

| | Return Registers: |
|---|---|
| 04h *unknown* | |
| *others, if any, unknown* | |
| | *unknown* |
| 05h get swap directory | |
| | CF clear |
| | CF clear<br>DX:AX -> ASCIZ swap directory name |

| Registers at call: | Return Registers: |
|---|---|
| AX = 5758h | |
| BL = function | |
| 06h *unknown* | CF clear |
| DX = *unknown* | |
| 07h *unknown* | |
| 08h *unknown* | |
| 09h get current application | BX = application number |
| 0Ah *unknown* | *unknown* |
| DX = application number | |
| DS:SI = *unknown* | |
| 0Bh *unknown* | |
| 0Ch *unknown* | *unknown* |
| DX = application number | |
| *others, if any, unknown* | |

| Registers at call: | Return Registers: |
|---|---|
| AX = 5758h | |
| BL = function | |
| 0Dh *unknown* | *unknown* |
| DX = application number | |
| *unknown* | |
| 0Eh get *unknown* | CF clear<br>AX = *unknown* |
| 0Fh set *unknown* flag | |
| 10h clear *unknown* flag | |
| 11h find application by name | CF clear |
| DS:SI -> ASCIZ application name | AX = application number or FFFFh if not loaded |
| 12h *unknown* | CF clear |
| DX = application number | *others, if any, unknown* |

**Registers at call:**
AX = 5758h
BL = function
- 13h *unknown*
- 14h *unknown*
  - same as function 13h
- 15h set *unknown*
  - DX = *unknown*
- 16h get *unknown*

## INT 21h Function 5758h

17h get *unknown*

18h BUG: branches incorrectly due to fencepost error

**Return Registers:**

CF clear

AX = *unknown* set by function 15h

BX = *unknown*
CX = *unknown* (may be pointer in BX:CX)

**See also:** Function 4C57h, Netroom Function 5757h (Chapter 41)

## OmniView

OmniView and its follow-on TaskView are TopView API-compatible DOS multitaskers by Sunny Hill Software. The callouts described in this section are invoked by OmniView to notify programs loaded before OmniView of state changes inside OmniView.

---

### INT 15h Function 5400h
### *INSTALLATION NOTIFICATION*

*Omniview Multitasker Callout*

**Registers at call:**
AX = 5400h
ES:BX -> device information tables
DI:DX -> dispatcher entry point

**Return Registers:**
none

**See also:** Function 5407h, DESQview XDI INT 2Fh Function DE00h (Chapter 44)

---

### INT 15h Function 5401h
### *PROCESS CREATION*

*Omniview Multitasker Callout*

**Registers at call:**
AX = 5401h
ES:BX = process handle

**Return Registers:**
none

**See also:** Function 5402h, DESQview XDI INT 2Fh Function DE04h (Chapter 44)

---

### INT 15h Function 5402h
### *PROCESS DESTRUCTION*

*Omniview Multitasker Callout*

**Registers at call:**
AX = 5402h
ES:DX = process handle

**Return Registers:**
none

**See also:** Function 5401h, DESQview XDI INT 2Fh Function DE05h (Chapter 44)

---

### INT 15h Function 5403h
### *SAVE*

*Omniview Multitasker Callout*

**Registers at call:**
AX = 5403h
ES:DX = process swapping out

**Return Registers:**
none

**See also:** Function 5404h, DESQview XDI INT 2Fh Function DE08h (Chapter 44)

---

### INT 15h Function 5404h
### *RESTORE*

*Omniview Multitasker Callout*

**Registers at call:**
AX = 5404h
ES:DX = process swapping in

**Return Registers:**
none

**See also:** Function 5403h, DESQview XDI INT 2Fh Function DE09h (Chapter 44)

### INT 15h Function 5405h
### *SWITCHING TO BACKGROUND*

*Omniview Multitasker Callout*

**Registers at call:**
AX = 5405h
ES:DX = process swapping in

**Return Registers:**
none

**See also:** Function 5406h

### INT 15h Function 5406h
### *SWITCHING TO FOREGROUND*

*Omniview Multitasker Callout*

**Registers at call:**
AX = 5406h
ES:DX = process swapping in

**Return Registers:**
none

**See also:** Function 5405h

### INT 15h Function 5407h
### *EXIT NOTIFICATION*

*Omniview Multitasker Callout*

**Registers at call:**
AX = 5407h

**Return Registers:**
none

**See also:** Function 5400h, DESQview XDI INT 2Fh Function DE03h (Chapter 44)

## PC-MOS/386

PC-MOS/386 is a multitasking/multiuser MS-DOS-compatible operating system by The Software Link, Inc. Version 5.01 is MS-DOS 5.0 compatible.

All of the PC-MOS/386 INT D4h functions conflict with STSC APL*PLUS/PC (Chapter 53) and IBM ROM BASIC (Chapter 54).

### INT 14h Function 05h
### *CHANGE PORT PROTOCOL*

*PC-MOS/386 v5.01 $serial.sys v5.04*

**Registers at call:**
AH = 05h
AL = protocol options (see Table 49-12)
BH = new XOFF character
BL = new XON character
DX = port number

**Return Registers:**
AH = FFh if invalid protocol

**Conflicts:** BIOS (Chapter 12), Digiboard (Chapter 12), FOSSIL (Chapter 12), MBBIOS (Chapter 12), MultiDOS Plus (Chapter 45)

**See also:** Function 06h, BIOS Functions 00h and 04h (Chapter 12)

**49-12 Bitfields for protocol options:**

| | |
|---|---|
| bit 7 | set to enable/disable CD monitoring, clear to set protocol |
| ---bit 7 set--- | |
| bit 5 | automatic restart enabled |
| bit 4 | CD monitoring enabled |
| ---bit 7 clear--- | |
| bit 3 | RTS/CTS |
| bit 2 | DTR/DSR |
| bit 1 | transmit XON/XOFF |
| bit 0 | receive XON/XOFF |

### INT 14h Function 06h
### *DRIVER 'ID' FUNCTION*

*PC-MOS/386 v5.01 $serial.sys v5.04*

**Registers at call:**
AH = 06h
DX = port number

**Return Registers:**
AH bit 7 set
AL = number of highest function supported by driver

**Conflicts:** FOSSIL (Chapter 12), MBBIOS (Chapter 12), MultiDOS Plus (Chapter 45), TelAPI (NI)
**See also:** Function 18h

---

### INT 14h Function 07h
*PC-MOS/386 v5.01 $serial.sys v5.04*

***SEND RS-232 BREAK***

**Registers at call:**
AH = 07h
BX = duration of break in clock ticks
DX = port number

**Return Registers:**
none

**Conflicts:** FOSSIL (Chapter 12), MBBIOS (Chapter 12), MultiDOS Plus (Chapter 45), TelAPI (NI)

---

### INT 14h Function 08h
*PC-MOS/386 v5.01 $serial.sys v5.04*

***INPUT STATUS CHECK***

**Registers at call:**
AH = 08h
DX = port number

**Return Registers:**
CF set if carrier loss detected
ZF set if input buffer empty
ZF clear if characters available
  AL = next character dequeued

**Conflicts:** Digiboard (Chapter 12), FOSSIL (Chapter 12), MBBIOS (Chapter 12), MultiDOS Plus (Chapter 45)

---

### INT 14h Function 09h
*PC-MOS/386 v5.01 $serial.sys v5.04*

***RESET I/O BUFFER POINTERS***

**Registers at call:**
AH = 09h
DX = port number

**Return Registers:**
none

**Conflicts:** Digiboard (Chapter 12), FOSSIL (Chapter 12), MBBIOS (Chapter 12), MultiDOS Plus (Chapter 45)
**See also:** Function 13h

---

### INT 14h Function 11h
*PC-MOS/386 v5.01 $serial.sys v5.04*

***DISABLE PORT***

**Registers at call:**
AH = 11h
DX = port number

**Return Registers:**
AL = status
  00h successful
  01h IRQ for port is shared
  02h IRQ was reserved

**Conflicts:** Digiboard (Chapter 12), FOSSIL (Chapter 12)
**See also:** Function 12h, BIOS Functions 04h and 05h

---

### INT 14h Function 12h
*PC-MOS/386 v5.01 $serial.sys v5.04*

***GET CURRENT PORT PARAMETERS***

**Registers at call:**
AH = 12h
DX = port number

**Return Registers:**
AH = status
  FFh port number invalid
AL = line parameters (see BIOS Function 00h, Chapter 12)
AH = flow control configuration (see Table 49-12)
CX:BX = bps rate
DL = XOFF character or 00h for none
DH = XON character or 00h for none

**Conflicts:** Digiboard (Chapter 12), FOSSIL (Chapter 12)

## INT 14h Function 13h
### REGISTER A PORT WITH A TERMINAL

*PC-MOS/386 v5.01 $serial.sys v5.04*

**Registers at call:**
AH = 13h
DX = port number

**Return Registers:**
AH = status
FFh port number invalid
else
ES:BX -> BYTE flag (00h buffer empty, FFh buffer contains data)

**Conflicts:** FOSSIL (Chapter 12)
**See also:** Function 17h

## INT 14h Function 14h
### OUTPUT STRING

*PC-MOS/386 v5.01 $serial.sys v5.04*

**Registers at call:**
AH = 14h
CX = number of characters in string
DX = port number
ES:BX -> string to be sent
SI = timeout in timer ticks or 0000h for default

**Return Registers:**
AX = number of bytes actually sent
ZF clear if successful
ZF set on timeout

**Conflicts:** FOSSIL (Chapter 12)
**See also:** Function 15h, BIOS Function 01h (Chapter 12)

## INT 14h Function 15h
### INPUT STRING

*PC-MOS/386 v5.01 $serial.sys v5.04*

**Registers at call:**
AH = 15h
CX = size of buffer
DX = port number
ES:BX -> buffer for received characters
SI = timeout in clock ticks or 0000h for default

**Return Registers:**
AX = number of characters actually read
ZF set on timeout (no data available)

**Conflicts:** FOSSIL (Chapter 12)
**See also:** Functions 14h and 16h, BIOS Function 02h (Chapter 12)

## INT 14h Function 16h
### LINK TO ANOTHER SERIAL DRIVER

*PC-MOS/386 v5.01 $serial.sys v5.04*

**Registers at call:**
AH = 16h
ES:BX -> calling driver's INT 14h entry point

**Return Registers:**
none

**Conflicts:** Digiboard (Chapter 12), FOSSIL (Chapter 12)

## INT 14h Function 17h
### WRITE MODEM CONTROL REGISTER

*PC-MOS/386 v5.01 $serial.sys v5.04*

**Registers at call:**
AH = 17h
AL = new value for UART's modem control register
DX = port number

**Return Registers:**
none

**Conflicts:** FOSSIL (Chapter 12)

## INT 14h Function 18h
### GET DRIVER DESCRIPTION

*PC-MOS/386 v5.01 $serial.sys v5.04*

**Registers at call:**
AH = 18h
DX = port number

**Return Registers:**
DS:BX -> 40-byte buffer containing a string identifying the serial driver

**Conflicts:** Digiboard (Chapter 12), FOSSIL (Chapter 12)
**See also:** Function 06h

---

### INT 14h Function 19h
*PC-MOS/386 v5.01 $serial.sys v5.04*
***SELECTIVE BUFFER FLUSH***

| Registers at call: | Return Registers: |
|---|---|
| AH = 19h | none |
| AL = what to flush | |
| bit 0: input buffer | |
| bit 1: output buffer | |
| DX = port number | |

**Conflicts:** Digiboard (Chapter 12), FOSSIL (Chapter 12)
**See also:** Function 09h

---

### INT 14h Function 20h
*PC-MOS/386 v5.01 $serial.sys v5.04*
***CHECK OUTPUT QUEUE***

| Registers at call: | Return Registers: |
|---|---|
| AH = 20h | AX = number of bytes in output buffer |
| DX = port number | |

**Conflicts:** FOSSIL (Chapter 12), MultiDOS Plus (Chapter 45), Alloy MW386 (NI)
**See also:** Function 0Ah

---

### INT 14h Function 21h
*PC-MOS/386 v5.01 $serial.sys v5.04*
***OUTPUT CHARACTER, WITH TIMEOUT***

| Registers at call: | Return Registers: |
|---|---|
| AH = 21h | none |
| AL = char to send | |
| DX = port number | |
| SI = timeout in timer ticks (0000h = default) | |

**Conflicts:** FOSSIL (Chapter 12), MultiDOS Plus (Chapter 45), Alloy MW386 (NI)
**See also:** Functions 0Eh and 22h, BIOS Function 01h (Chapter 12)

---

### INT 14h Function 22h
*PC-MOS/386 v5.01 $serial.sys v5.04*
***RECEIVE CHARACTER, WITH TIMEOUT***

| Registers at call: | Return Registers: |
|---|---|
| AH = 22h | AH = port status (see Function 03h bits 0..6) |
| DX = port number | bit 7 set indicates time-out |
| SI = timeout in timer ticks (0000h = default) | AL = character received |

**Conflicts:** MultiDOS Plus (Chapter 45), Alloy MW386 (NI)
**See also:** Functions 0Fh and 21h, BIOS Function 02h (Chapter 12)

---

### INT 14h Function 23h
*PC-MOS/386 v5.01 $serial.sys v5.04*
***DECLARE PORT OWNERSHIP***

| Registers at call: | Return Registers: |
|---|---|
| AH = 23h | none |
| DX = port number | |
| BX = TCB segment/selector address of owner task | |

**Conflicts:** MultiDOS Plus (Chapter 45), Alloy MW386 (NI)
**See also:** Function 0Dh

---

### INT 14h Function 24h
*PC-MOS/386 v5.01 $serial.sys v5.04*
*UNDETERMINED FUNCTION*

| Registers at call: | Return Registers: |
|---|---|
| AH = 24h | *unknown* |

**Conflicts:** MultiDOS Plus (Chapter 45), Alloy MW386 (NI)

### INT 17h Function FFh
*PC-MOS/386 v5.01 (undocumented)*

### *PRINT SPOOLER - CLOSE SPOOL FILE*

**Purpose:** Close the spool file immediately instead of waiting for the close time to elapse.

**Registers at call:**
AH = FFh
BX = 0000h
CX = 0000h
DX = printer port number

**Return Registers:**
AH = printer status (see Table 15-1)

**See also:** Function 03h, BIOS Function 01h (Chapter 15)

### INT 21h Function 3000h, Subfunction 3000h
*PC-MOS/386 v3.0*

### *INSTALLATION CHECK/GET VERSION*

**Registers at call:**
AX = 3000h
BX = 3000h
CX = 3000h
DX = 3000h

**Return Registers:**
AX = PC-MOS version

**Conflicts:** MS-DOS (Chapter 24)

**See also:** INT D4h Functions 02h and 10h, MS-DOS Function 30h (Chapter 24)

### INT 38h
*PC-MOS/386 v3.0*

### *API*

**Details:** This API was moved to INT D4h sometime between versions 3.0 and 5.01; v3.0 supported at least functions 02h,04h,0703h,10h,11h, and 12h.

**Conflict:** Floating-Point emulation (Chapter 54)

**See also:** INT D4h Functions 02h, 04h, 07h, 10h, and 11h

### INT D4h Functions 00h and 01h
*PC-MOS/386 v5.01*

### *OBSOLETE FUNCTIONS*

**Purpose:** PC-MOS/386 v5.01 reports that these functions are no longer supported and enters an endless loop.

**Registers at call:**
AH = 00h and 01h

**Return Registers:**
none

**Conflicts:** STSC APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54)

### INT D4h Function 02h
*PC-MOS/386 v3.0+ Obsolete*

### *GET SYSTEM CONTROL BLOCK POINTER*

**Registers at call:**
AH = 02h

**Return Registers:**
AX = 0000h
ES:BX -> System Control Block in V86 mode (see Table 49-13)
ES:EBX -> System Control Block in native mode (see Table 49-13)

**Details:** This function has been superseded by Function 26h.

**See also:** Functions 04h, 10h, 26h, 28h, 29h, and 2Ah; INT 21h Function 3000h; INT 38h

**49-13 Format of System Control Block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | pointer to first TCB in chain |
| 02h | 17 BYTEs | reserved |
| 13h | WORD | pointer to current task's TCB |
| 15h | WORD | pointer to TCB of visible (console) task |

### INT D4h Function 03h
*PC-MOS/386 v5.01*

### *GET/SET EXTENDED DIRECTORY INFORMATION*

**Registers at call:**
AH = 03h

AL = subfunction (00h get, 01h set)
DS:(E)DX -> pathname
ES:(E)BX -> 10-byte buffer for directory information (see Table 49-14)

**Return Registers:**
CF clear if successful
AL = permitted access level for file (00h-03h)
ES:(E)BX -> modified buffer (if AL=01h on entry)
CF set on error
AX = error code (see Table 24-2)

**Details:** BX/DX are used in V86 mode, EBX/EDX in native mode.

The file class cannot be changed for files because it affects the encryption method, but directories can have their classes changed.

**49-14 Format of directory information:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | reserved (0) |
| 01h | BYTE | file class ('A'-'Z' or 00h) |
| 02h | DWORD | user ID of file creator |
| 06h | WORD | file creation time (see Table 24-!I) |
| 08h | WORD | file creation date (see Table 24-!J) |

## INT D4h Function 04h

*PC-MOS/386 v3.0+*
*Obsolete*

### *GET TASK CONTROL BLOCK*

**Registers at call:**
AH = 04h
BX = task ID or FFFFh for calling task

**Return Registers:**
CF clear if successful
ES = segment of Task Control Block (TCB) (see Table 49-15)
CF set on error
AX = error code (see Table 24-2)

**Details:** This function has been superseded by Function 27h.

**See also:** Functions 02h, 27h, 28h, 29h, and 2Ah; INT 38h

**49-15 Format of Task Control Block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | signature byte "H" if allocated from system memory pool |
| 01h | BYTE | header block ID<br>"B" = block device block (see Table 49-22)<br>"C" = current directory block (see Table 49-21)<br>"F" = task file block (see Table 49-18)<br>"G" = global file block (see Table 49-20)<br>"R" = record lock block (see Table 49-23)<br>"T" = TCB |
| 02h | WORD | length of block in paragraphs |
| 04h | WORD | segment address of next header block (0000h if last) |
| 06h | WORD | segment address of previous header block (0000h if first) |
| 08h | WORD | pointer to next TCB |
| 0Ah | WORD | pointer to previous TCB |
| 0Ch | WORD | pointer to associated TCB (if applicable) |
| 0Eh | WORD | reserved |
| ---TCB--- | | |
| 10h | WORD | TCB task ID |
| 12h | WORD | native context save area |
| 14h | WORD | start address of task |
| 16h | WORD | end address of task |
| 18h | BYTE | task priority |
| 19h | BYTE | task time slice |
| 1Ah | BYTE | "TCBWAIT" run status of task |

| | | |
|---|---|---|
| 1Bh | BYTE | "TCBSTAT" what the task is waiting for |
| 1Ch | DWORD | address of polling routine |
| 20h | BYTE | error code from last function call |
| 21h | 11 BYTEs | name of currently executing task |
| 2Ch 4 | BYTEs | *unknown* |
| 30h | BYTE | keyboard disabled if bit 1 set |
| 31h | BYTE | current shift state and toggles |
| 32h | 2 BYTEs | *unknown* |
| 34h | BYTE | current video mode |
| 35h | BYTE | current video page |
| 36h | BYTE | number of text columns per screen |
| 37h | BYTE | number of text rows per screen |
| 38h | WORD | length of video buffer |
| 3Ah | WORD | video page length |
| 3Ch | WORD | apge start address in video RAM |
| 3Eh 4 | WORDs | current cursor positions for four screen pages |
| 46h | 8 BYTEs | *unknown* |
| 4Eh | WORD | current cursor type |
| 50h | BYTE | current palette setting |
| 51h | BYTE | original video mode |
| 52h | BYTE | start CRT row (00h or 01h) |
| 53h | BYTE | video RAM in task active |
| 54h | WORD | handle of video save area |
| 56h | WORD | page count of video save area |
| 58h | WORD | segment address of video save area |
| 5Ah | WORD | poitner to first Task File Block (see Table 49-18) |
| 5Ch | WORD | pointer to first Current Directory Block (see Table 49-21) |
| 5Eh | WORD | pointer to active Current Directory Block (see Table 49-21) |
| 60h | BYTE | number of drives |
| 61h | BYTE | current drive (0=A:, etc.) |
| 62h | DWORD | disk transfer address |
| 66h | 4 BYTEs | *unknown* |
| 6Ah | BYTE | verify flag (nonzero = on) |
| 6Bh | BYTE | break flag (nonzero = on) |
| 6Ch | WORD | share/lock retry count |
| 6Eh | WORD | ticks between share/lock retries |
| 70h | BYTE | remote printer flags (see Table 49-16) |
| 71h | BYTE | ETX/ACK delay count |
| 72h | WORD | spooler segment address |
| 74h | 2 BYTEs | *unknown* |
| 76h | 3 BYTEs | remote printer redirection for LPT1 through LPT3 (see Table 49-17) |
| 79h | 2 BYTEs | *unknown* |
| 7Bh | DWORD | offset of username in TCB |
| 7Fh | BYTE | current output class |
| 80h | 7 BYTEs | protection access rights, 2 bits per class (writeable!) |
| 87h | 122 BYTEs | *unknown* |
| 101h | BYTE | TCB sleep downcounter value |
| 102h | 20 BYTEs | *unknown* |
| 116h | BYTE | last scan code |

## INT D4h Function 04h

| | | |
|---|---|---|
| | ... | |
| 5D0h | DWORD | far pointer to Device Driver Terminal's entry point |
| 5D4h | WORD | offset of logical screen |
| 5D6h | WORD | segment of logical screen |
| 5D8h | WORD | cursor offset within page |
| 5DAh | BYTE | screen columns |
| 5DBh | WORD | async port number (0000h = none) |
| 5DDh | DWORD | physical baudrate |
| 5E1h | 19 BYTEs | reserved for Device Driver Terminal (DDT) |
| | ... | |
| 7A6h | DWORD | far pointer to unregister calling chain |

**49-16 Bitfield for remote printer flags:**

| | |
|---|---|
| bit 0 | LPT1 to terminal |
| bit 1 | LPT2 to terminal |
| bit 2 | LPT3 to terminal |
| bit 3 | escape to printer pending |
| bit 4 | use XON/XOFF |
| bit 5 | use ETX/ACK |
| bit 6 | waiting for ACK or XON |
| bit 7 | transparent printing on |

**49-17 Values for remote printer redirection:**

| | |
|---|---|
| 00h | not redirected |
| 01h | redirected to COM1 |
| ... | |
| 18h | redirected to COM24 |
| 51h | redirected to LPT1 |
| 52h | redirected to LPT2 |
| 53h | redirected to LPT3 |

**49-18 Format of Task File Block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 16 BYTEs | common header (see Table 49-15) |
| 10h | WORD | segment address of next TFB |
| 12h | WORD | segment address of previous TFB |
| 14h | WORD | segment address of TFB's Global File Block (see Table 49-20) |
| 16h | WORD | segment address of owner's PSP |
| 18h | WORD | file handle |
| 1Ah | 3 BYTEs | *unknown* |
| 1Dh | DWORD | file position |
| 21h | 4 BYTEs | *unknown* |
| 25h | BYTE | IOCTL flags (see Table 49-19) |
| 26h | 2 BYTEs | *unknown* |

**49-19 Bitfields for IOCTL flags:**

| | |
|---|---|
| bit 0 | stdin |
| bit 1 | stdout |
| bit 2 | null device |
| bit 3 | clock device |
| bit 4 | reserved |
| bit 5 | ASCII mode instead of binary |
| bit 6 | EOF encountered on input |
| bit 7 | device rather than file |

### 49-20 Format of Global File Block:

| Offset | Size | Description |
|---|---|---|
| 00h | 16 BYTEs | common header (see Table 49-15) |
| 10h | 10 BYTEs | *unknown* |
| 1Ah | WORD | file attribute |
| 1Ch | BYTE | *unknown* |
| 1Dh | DWORD | address of device driver |
| 21h | WORD | first cluster |
| 23h | WORD | time of last modification |
| 25h | WORD | date of last modification |
| 27h | DWORD | size of file in bytes |
| 2Bh | 11 BYTEs | *unknown* |
| 36h | 11 BYTEs | device name or FCB-format filename |
| 41h | WORD | segment addres off TFB list |
| 43h | WORD | segment address of first RLB (see Table 49-23) (0000h = none) |
| 45h | BYTE | flag: nonzero if GFB refers to character device |
| 46h | WORD | address of Block Device Block (see Table 49-22) |
| 48h | WORD | sector of file's directory entry |
| 4Ah | WORD | high word of file's directory entry |
| 4Ch | WORD | ofsset of directory entry within sector |

### 49-21 Format of Current Directory Block:

| Offset | Size | Description |
|---|---|---|
| 00h | 16 BYTEs | common header (see Table 49-15) |
| 10h | BYTE | drive number |
| 11h | BYTE | *unknown* |
| 12h | 64 BYTEs | directory name |
| 52h | WORD | first directory cluster (0000h = root) |

### 49-22 Format of Block Device Block:

| Offset | Size | Description |
|---|---|---|
| 00h | 16 BYTEs | common header (see Table 49-15) |
| 10h | BYTE | logical drive |
| 11h | BYTE | unit passed to driver |
| 12h | WORD | sector size |
| 14h | BYTE | cluster mask |
| 15h | BYTE | cluster shift count |
| 16h | WORD | starting sector of first FAT |
| 18h | BYTE | number of FATs |
| 19h | WORD | number of root directories |
| 1Bh | WORD | sector number of first data sector (cluster 0002h) |
| 1Dh | WORD | number of clusters + 1 (number of highest data cluster) |
| 1Fh | BYTE | number of sectors in FAT |
| 20h | WORD | beginning root directory sector number |
| 22h | DWORD | device driver address |
| 26h | BYTE | media descriptor byte |
| 27h | 5 BYTEs | *unknown* |
| 2Ch | BYTE | flag: volume > 32MB |
| 2Dh | BYTE | *unknown* |
| 2Eh | BYTE | number of sectors per cluster |
| 2Fh | WORD | number of clusters on device |
| 31h | WORD | number of free clusters (FFFFh = unknown) |

## INT D4h Function 04h

| | | |
|---|---|---|
| 33h | WORD | root directory cluster number |
| 35h | WORD | pointer to alias/subst string |
| 37h | WORD | TCB segment address of owner (0000h = none) |

**49-23 Format of Record Lock Block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 16 BYTEs | common header (see Table 49-15) |
| 10h | WORD | segment address of owner's PSP |
| 12h | WORD | segment address of Global File Block (see Table 49-20) |
| 14h | WORD | segment address of owner's Task File Block (see Table 49-18) |
| 16h | DWORD | file offset of locked region start |
| 1Ah | DWORD | length of locked region |
| 1Eh | WORD | owner's handle for file |

## INT D4h Function 07h
*PC-MOS/386 v3.0+*
### *WAIT FOR EVENT*

**Registers at call:**
AH = 07h
AL = events to monitor (see Table 49-24)
BX = number of timer ticks until timeout if AL bit 1 set
CX = bitmap of IRQs to monitor if AL bit 2 set (bit 0 = IRQ0 .. bit 15 = IRQ15)
DX = port to monitor if AL bit 3 set

**Return Registers:**
CF clear if successful
AL = type of event which woke up task (see Table 49-24)
CX = IRQ (if any) which awakened task
DX = port (if any) which awakened task
CF set on error
AX = error code (see Table 24-2)

**Details:** A device driver may make this call with AL=00h, which indicates that the driver is responsible for setting and clearing the TCBWAIT field in the TCB. To put task to sleep, set TCBWAIT bits 2-0 to 001; to reawaken it, set bit 1 (leaving other bits unchanged).
**Conflicts:** STSC APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54)
**See also:** Function 04h, INT 38h, BIOS INT 16h Function 00h (Chapter 4)

**49-24 Bitfields for events to monitor:**

| | |
|---|---|
| bit 0 | keystroke |
| bit 1 | timeout |
| bit 2 | IRQ |
| bit 3 | port access |
| bit 7 | return status of user poll routine (other bits ignored if set) |

## INT D4h Function 10h
*PC-MOS/386 v3.0+*
### *ENTER/LEAVE NATIVE 386 EXECUTION MODE*

**Registers at call:**
AH = 10h
AL = direction (00h return to V86 mode, 01h enter native mode)
CX = length in bytes of Native Context Area ( >=1024 )
DX = segment of Native Context Area

**Return Registers:**
CF clear if successful
running in desired mode at instruction following INT D4h call
all segment registers converted to appropriate selectors/segments
CF set on error
AX = error code (see Table 24-2)

**Details:** MS-DOS calls are available in protected mode.
**See also:** Functions 11h, 12h, and 13h; INT 38h; DPMI INT 2Fh Function 1687h (Chapter 30); VCPI INT 67h Function DE0Ch (Chapter 29)

### INT D4h Function 11h
### *ALLOCATE NATIVE MODE MEMORY BLOCK*

*PC-MOS/386 v3.0+*

**Registers at call:**
AH = 11h
EBX = block length in bytes

**Return Registers:**
CF clear if successful
EBX = number of bytes actually allocated
ES = selector for allocated block
CF set on error
AX = error code (see Table 24-2)

**Details:** The memory must be released before the program terminates.

**See also:** Functions 10h and 12h, INT 38h

### INT D4h Function 12h
### *FREE NATIVE MODE MEMORY BLOCK*

*PC-MOS/386 v3.0+*

**Registers at call:**
AH = 12h
ES = selector for block to free

**Return Registers:**
CF clear if successful
CF set on error
AX = error code (see Table 24-2)

**See also:** Functions 10h, 11h, and 13h; INT 38h

### INT D4h Function 13h
### *GET ALIAS FOR SELECTOR*

*PC-MOS/386 v5.01*
*Native mode only*

**Registers at call:**
AH = 13h
AL = type of alias selector (00h data, 01h stack, 02h code)
BX = selector

**Return Registers:**
CF clear if successful
AX = new selector or 0000h if BX selector not found
CF set on error
AX = error code (see Table 24-2)

**See also:** Functions 11h and 12h

### INT D4h Function 16h
### *SET/CLEAR IRQ RESERVATION*

*PC-MOS/386 v5.01*

**Registers at call:**
AH = 16h
AL = function (00h clear, 01h set reservation)
CX = IRQ number

**Return Registers:**
AX = status
0000h successful
0001h currently reserved by another task

**See also:** Function 07h, INT 14h Function 11h

### INT D4h Function 19h
### *GET TASK ID*

*PC-MOS/386 v5.01*

**Registers at call:**
AH = 19h

**Return Registers:**
BX = caller's task ID

**Conflicts:** STSC APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54)

**See also:** Functions 1Dh and 1Eh

### INT D4h Function 1Ah
### *GET/SET TASK PRIORITY*

*PC-MOS/386 v5.01*

**Registers at call:**
AH = 1Ah
AL = subfunction (00h read, 01h set, 02h get and set)
BX = task ID (FFFFh for current task)
CL = new priority value

**Return Registers:**
CF clear if successful
CL = current priority value
CF set on error
AX = error code (see Table 24-2)

**See also:** Functions 1Bh and 1Ch

---

**INT D4h Function 1Bh** *PC-MOS/386 v5.01*

***GET/SET TIME SLICE***

**Registers at call:**
AH = 1Bh
AL = subfunction (00h read, 01h set, 02h get and set)
BX = task ID (FFFFh for current task) (see Function 19h)
CL = new time slice value

**Return Registers:**
CF clear if successful
CL = current time slice value
CF set on error
AX = error code (see Table 24-2)

**See also:** Functions 1Ah and 1Ch

---

**INT D4h Function 1Ch** *PC-MOS/386 v5.01*

***GET/SET KEYBOARD MODE***

**Registers at call:**
AH = 1Ch
AL = subfunction (00h enable, 01h disable, 02h get mode)
BX = task ID (FFFFh for current task)

**Return Registers:**
CF clear if successful
CL = current keyboard state
CF set on error
AX = error code (see Table 24-2)

**See also:** Functions 1Ah and 1Bh

---

**INT D4h Function 1Dh** *PC-MOS/386 v5.01*

***GET CURRENT PROGRAM NAME***

**Registers at call:**
AH = 1Dh
BX = task ID (FFFFh for current task) (see Function 19h)
ES:DI -> buffer for program name (see see Table 49-25)

**Return Registers:**
CF clear if successful
ES:DI buffer filled
CF set on error
AX = error code (see Table 24-2)

**See also:** Functions 19h and 1Eh

**49-25 Format of buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 8 BYTEs | filename |
| 08h | 3 BYTEs | extension |

---

**INT D4h Function 1Eh** *PC-MOS/386 v5.01*

***GET CURRENT USERNAME AND SECURITY CLASS***

**Registers at call:**
AH = 1Eh
BX = task ID (FFFFh for current task)
ES:DI -> 4-byte buffer for username

**Return Registers:**
CF clear if successful
CL = security class
20h (' ') none
41h-5Ah ('A'-'Z') security level
ES:DI buffer filled
CF set on error
AX = error code (see Table 24-2)

**See also:** Functions 19h and 1Dh

---

**INT D4h Function 1Fh** *PC-MOS/386 v5.01*

***GET TASK PARTITION INFORMATION***

**Registers at call:**
AH = 1Fh
BX = task ID (FFFFh for current task) (see Function 19h)

**Return Registers:**
CF clear if successful
CX = start segment of task
DX = ending segment of task
CF set on error
AX = error code (see Table 24-2)

**See also:** Function 2Dh

---

### INT D4h Function 20h

PC-MOS/386 v5.01

***GET PORT AND BAUDRATE INFORMATION***

**Registers at call:**
AH = 20h
BX = task ID (FFFFh for current task) (see Function 19h)

**Return Registers:**
CF clear if successful
  CX = port number (0000h if none)
  DI:SI = baudrate (if CX nonzero)
CF set on error
  AX = error code (see Table 24-2)

**Conflicts:** STSC APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54)
**See also:** FOSSIL INT 14h Function 0Ch (Chapter 12)

---

### INT D4h Function 21h

PC-MOS/386 v5.01

***REMOVE A TASK***

**Registers at call:**
AH = 21h
BX = task ID (FFFFh for current task) (see Function 19h)

**Return Registers:**
CF clear if successful
  AX = ASCII percentage of System Memory Pool use (AH = tens digit, AL = ones digit)
  DS,SI destroyed
CF set on error
  AX = error code (see Table 24-2)

**See also:** Function 22h

---

### INT D4h Function 22h

PC-MOS/386 v5.01

***ADD A TASK TO THE SYSTEM***

**Registers at call:**
AH = 22h
DS:SI -> addtask data structure (see Table 49-27)

**Return Registers:**
CF clear if successful
  ES = segment address of the new task's TCB data structure
CF set on error
  AX = error code (see Table 49-26)

**See also:** Function 21h

**49-26 Values for error code:**

| | |
|---|---|
| 08h | insufficient memory |
| 0Bh | invalid addtask structure format |
| 12h | insufficient available space in system memory pool |
| 1Fh | general failure |
| 55h | already allocated |
| 57h | if task already in use or invalid parameter |

**49-27 Format of addtask data structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | task size in KB (min 16KB) |
| 02h | WORD | task ID (0000h for automatic selection) |
| 04h | BYTE | task class (' ' or 'A'-'Z') |
| 05h | DWORD | -> ASCIZ name of task startup batchfile |
| 09h | DWORD | -> task's terminal driver (0000000h = background task) |
| 0Dh | WORD | task port |
| 0Fh | DWORD | task baud rate |
| 13h | DWORD | (return) total extended memory |
| 17h | DWORD | (return) number of 4K extended memory pages allocated |
| 1Bh | WORD | (return) paragraphs of system memory pool allocated |
| 1Dh | WORD | (return) system memory pool size in paragraphs |
| 1Fh | WORD | (return) ASCII task percentage of system memory pool |
| 21h 3 | BYTEs | reserved |

### INT D4h Function 23h

*PC-MOS/386 v5.01*

### *CHANGE TERMINAL DRIVER*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 23h | CF clear if successful |
| BX = task ID (FFFFh for current task) | CF set on error |
| DS:SI -> entry point of the new Device Driver Terminal | AX = error code (see Table 24-2) |

### INT D4h Function 24h

*PC-MOS/386 v5.01*
*(undocumented)*

### *GET OPERATING SYSTEM SERIAL NUMBER*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 24h | DS:DX -> '$'-terminated string containing the serial number |

### INT D4h Function 25h

*PC-MOS/386 v5.01*

### *IDENTIFY LOAD ADDRESS OF DEVICE DRIVER*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 25h | AX = segment address of driver in system memory pool |
| DX = driver's CS value | (0000h if the driver is not within the system memory pool) |

### INT D4h Function 26h

*PC-MOS/386 v5.01*

### *GET SYSTEM CONTROL BLOCK SEGMENT/SELECTOR*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 26h | DX = segment/selector of the System Control Block (see Table 49-13) |

**Details:** This function supersedes Function 02h.

**See also:** Functions 02h, 27h, 28h, 29h, and 2Ah

### INT D4h Function 27h

*PC-MOS/386 v5.01*

### *GET TASK CONTROL BLOCK SEGMENT/SELECTOR*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 27h | CF clear if successful |
| BX = task ID (FFFFh if current task) (see Function 19h) | DX = segment/selector for the Task Control Block (see Table 49-15) |
| | CF set on error |
| | AX = error code (see Table 24-2) |

**Details:** This function supersedes Function 04h.

**See also:** Functions 26h, 28h, 29h, and 2Ah

### INT D4h Function 28h

*PC-MOS/386 v5.01*

### *GET CONTROL BLOCK DATA FROM SCB OR TCB*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 28h | CF clear if successful |
| BX = offset into control block at which to start reading | CF set on error |
| | AX = error code (see also Table 24-2) |
| CX = number of bytes to read | 05h access denied due to an invalid segment/selector |
| DX = segment/selector of control block obtained via Functions 26h or 27h | |
| ES:DI -> buffer for data | |

**Conflicts:** STSC APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54)

**See also:** Functions 26h, 27h, 29h, and 2Ah

### INT D4h Function 29h

*PC-MOS/386 v5.01*

### *WRITE CONTROL BLOCK DATA INTO SCB OR TCB*

**Registers at call:**
AH = 29h
BX = offset into control block at which to start writing
CX = number of bytes to write

| | |
|---|---|
| DX = segment/selector of control block obtained via Functions 26h or 27h | **Return Registers:**<br>CF clear if successful |
| DS:SI -> buffer containing data to be written | CF set on errro<br>AX = error code (see also Table 24-2)<br>05h access denied due to an invalid segment/selector |

**Details:** This function performs no bounds checking.
**See also:** Functions 26h, 27h, 28h, and 2Ah

---

### INT D4h Function 2Ah
*PC-MOS/386 v5.01*
### *SWAP CONTROL BLOCK DATA OF SCB OR TCB*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 2Ah | CF clear if successful |
| BX = offset into control block at which to start swap | CF set on error<br>AX = error code (see also Table 24-2) |
| CX = number of bytes to swap | 05h access denied due to an invalid segment/selector |

DX = segment/selector of control block obtained via Functions 26h or 27h
DS:SI -> buffer containing new data and to receive current data

**Details:** Interrupts are disabled during the swap to prevent corruption.
**See also:** Functions 26h, 27h, 28h, and 29h

---

### INT D4h Function 2Ch
*PC-MOS/386 v5.01*
### *GET/SET SPOOLER PARAMETERS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 2Ch | CF clear if successful |
| AL = function (see Table 49-28) | CF set on error<br>AX = error code (see Table 24-2) |

**See also:** Functions 00h, 02h, and 03h

**49-28 Parameter function codes:**

| | |
|---|---|
| 00h | set spooler timeout<br>CX = timout value in seconds |
| 01h | get spooler timeout<br>Returns CX = current timeout in seconds |
| 02h | get spooler parameters<br>Returns CH = priority (00h-09h)<br>CL = disposition (d, h, i, n, s)<br>SI = class (a - z) |
| 03h | set spooler parameters<br>CH = priority (00h-09h)<br>CL = disposition (d, h, i, n, s)<br>SI = class (a - z)<br>BX = task ID (FFFFh for current task)<br>DX = LPT number |

---

### INT D4h Function 2Dh
*PC-MOS/386 v5.01*
### *GET MAXIMUM TASK SIZE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 2Dh | DX = maximum task size in paragraphs<br>BX = start address of task space |

**Conflicts:** STSC APL*PLUS/PC (Chapter 53), IBM ROM BASIC (Chapter 54)
**See also:** Functions 1Fh and 22h

## PC Tools CPTASK

CPTASK is a task switcher by Central Point Software included with PC Tools version 8 and higher.

### INT D4h Function 2Ah

All of the CPTASK functions listed in this section conflict with KBUF (Chapter 13) and the Optima/ET3000 Zooming TSR (Chapter 9).

---

### INT 16h Function FF80h, Subfunction 0000h

***INSTALLATION CHECK***

*PC Tools v8+ CPTASK*
*(undocumented)*

**Registers at call:**
AX = FF80h
BX = 0000h
CX = 0000h
DX = 0000h

**Return Registers:**
CX = 5555h if installed

---

### INT 16h Function FF80h, Subfunction 0001h

***GET*** *UNKNOWN VALUES*

*PC Tools v8+ CPTASK*
*(undocumented)*

**Registers at call:**
AX = FF80h
BX = 0001h
*unknown*

**Return Registers:**
DX:SI -> *task list*
BX = *unknown* (*PSP segment of resident code*)

---

### INT 16h Function FF80h, Subfunction 0002h

***GET*** *UNKNOWN POINTER*

*PC Tools v8+ CPTASK*
*(undocumented)*

**Registers at call:**
AX = FF80h
BX = 0002h

**Return Registers:**
DX:SI -> *unknown*

---

### INT 16h Function FF80h, Subfunction 0003h

***GET*** *UNKNOWN VALUE*

*PC Tools v8+ CPTASK*
*(undocumented)*

**Registers at call:**
AX = FF80h
BX = 0003h

**Return Registers:**
AX = *unknown*

---

### INT 16h Function FF80h, Subfunction 0004h

***SET*** *UNKNOWN VALUE*

*PC Tools v8+ CPTASK*
*(undocumented)*

**Registers at call:**
AX = FF80h
BX = 0004h
CX = new value of *unknown*

**Return Registers:**
*unknown*

**Details:** This function also sets the *unknown* flag.

---

### INT 16h Function FF80h, Subfunction 0005h

*GET NUMBER OF ACTIVE TASKS*

*PC Tools v8+ CPTASK*
*(undocumented)*

**Registers at call:**
AX = FF80h
BX = 0005h

**Return Registers:**
AX = *number of active tasks*

---

### INT 16h Function FF80h, Subfunction 0006h

***GET AND CLEAR*** *UNKNOWN* ***FLAG***

*PC Tools v8+ CPTASK*
*(undocumented)*

**Registers at call:**
AX = FF80h
BX = 0006h

**Return Registers:**
AX = old state (0000h clear, 0001h set)

---

### INT 16h Function FF80h, Subfunction 0007h

*UNDETERMINED FUNCTION*

*PC Tools v8+ CPTASK*
*(undocumented)*

**Registers at call:**
AX = FF80h
BX = 0007h

ES:DI -> *unknown*
*unknown*

**Return Registers:**
*unknown*

---

### INT 16h Function FF80h, Subfunction 0008h

*PC Tools v8+ CPTASK*
*(undocumented)*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = FF80h
BX = 0008h
*unknown*

**Return Registers:**
*unknown*

---

### INT 16h Function FF80h, Subfunction 0009h

*PC Tools v8+ CPTASK*
*(undocumented)*

***GET** UNKNOWN VALUES*

**Registers at call:**
AX = FF80h
BX = 0009h

**Return Registers:**
CL = *unknown*
CH = *unknown* (01h or 02h)

---

### INT 16h Function FF80h, Subfunction 4350h

*PC Tools v8+ CPTASK*
*(undocumented)*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = FF80h
BX = 4350h ('CP')
CX = 5354h ('ST')
*unknown*

**Return Registers:**
*unknown*

---

### INT 16h Function FF9Ch

*PC Tools v8+ CPTASK*
*(undocumented)*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = FF9Ch
BL = *unknown* (00h,01h,02h)
*unknown*

**Return Registers:**
*unknown*

**Details:** This function sounds a triple-length beep.

---

### INT 16h Function FF9Dh

*PC Tools v8+ CPTASK, VSAFE*
*(undocumented)*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = FF9Dh
ES:BX -> *unknown* word

**Return Registers:**
*unknown*

**Details:** If ES is non-zero, the word pointed at by ES:BX determines whether the *unknown* flag is cleared (word = 0000h) or set (word is nonzero). The flag is always cleared if ES=0000h.

## Software Carousel

Software Carousel is a task switcher by SoftLogic Solutions, Inc.

All of the calls described in this section conflict with Novell NetWare print services (see *Network Interrupts*).

---

### INT 21h Function DF00h

*Software Carousel*
*(undocumented)*

***INSTALLATION CHECK***

**Registers at call:**
AX = DF00h
DX = 534Ch ("SL")
DI = 534Ch ("SL")

**Return Registers:**
AX = 00FFh if installed
*other register, if any, unknown*

INT 16h Function FF80h, Subfunction 0008h

---

## INT 21h Function DF01h
*Software Carousel*

### *SWITCH TO ANOTHER TASK*

**Registers at call:**
AX = DF01h
BL = task number (00h = next task)

**Return Registers:**
AL = status
00h Carousel not running
01h successful
FFh unsucessful
AH = error code (01h,02h) (see Table 49-29)

---

**49-29 Values for error code:**

| | |
|---|---|
| 00h | invalid subfunction in AL |
| 01h | invalid task number |
| 02h | tried to switch to task with no memory size |
| 03h | tried to kill program in partition with no program running |
| 04h | tried to change size of an active memory partition |
| 05h | invalid memory size |
| 06h | tried to send command to task with a pending previous command |

---

## INT 21h Function DF02h
*Software Carousel*

### *KILL PROGRAM IN MEMORY PARTITION*

**Registers at call:**
AX = DF02h
BL = task number

**Return Registers:**
AL = status
00h Carousel not running
01h successful
FFh unsucessful
AH = error code (01h,03h) (see Table 49-29)

---

## INT 21h Function DF03h
*Software Carousel*

### *GET PARTITION SIZE AND PROGRAM STATUS*

**Registers at call:**
AX = DF03h
BL = task number

**Return Registers:**
AL = status
00h Carousel not running
01h successful
BL = partition state (00h no program running, 01h program running)
DX = partition size in KB
FFh unsucessful
AH = error code (01h) (see Table 49-29)

**See also:** Function DF05h

---

## INT 21h Function DF04h
*Software Carousel*

### *GET PARTITION NAME*

**Registers at call:**
AX = DF04h
BL = task number

**Return Registers:**
AL = status
00h Carousel not running
01h successful
CX = length of name (00h if default partition name)
ES:BX -> partition name (if CX nonzero)
FFh unsucessful
AH = error code (01h) (see Table 49-29)

**See also:** Function DF06h

## INT 21h Function DF05h
Software Carousel

### *CHANGE PARTITION SIZE*

**Registers at call:**
AX = DF05h
BL = task number
DX = new size in KB

**Return Registers:**
AL = status
  00h Carousel not running
  01h successful
  FFh unsucessful
    AH = error code (01h,04h,05h) (see Table 49-29)
BX = minimum size allowed
CX = maximum size available

**See also:** Function DF03h

## INT 21h Function DF06h
Software Carousel

### *CHANGE PARTITION NAME*

**Registers at call:**
AX = DF06h
BL = task number
CX = length of new name (00h to use default, max 18h)
DS:SI -> new name

**Return Registers:**
AL = status
  00h Carousel not running
  01h successful
  FFh unsucessful
    AH = error code (01h) (see Table 49-29)

**See also:** Function DF04h

## INT 21h Function DF07h
Software Carousel

### *SEND COMMAND TO MEMORY SECTION*

**Registers at call:**
AX = DF07h
BL = task number
CX = length of command (max 8 chars)
DS:SI -> command line

**Return Registers:**
AL = status
  00h Carousel not running
  01h successful
  FFh unsucessful
    AH = error code (01h,06h) (see Table 49-29)

**Details:** The maximum length seems too small and may be a typo in the documentation for 80 characters.

## INT 21h Function DF08h
Software Carousel

### *SELECTIVELY ENABLE/DISABLE MENU AND SWITCHING*

**Registers at call:**
AX = DF08h
BL = new state of keyboard (00h disabled, 01h enabled)

**Return Registers:**
AL = status
  00h Carousel not running
  01h successful

**Details:** When the keyboard is is disabled, the user may neither access the Carousel menu nor switch to another memory section.

## INT 21h Function DF09h
Software Carousel

### *BOOT THE SYSTEM*

**Registers at call:**
AX = DF09h

**Return Registers:**
AL = status
  00h Carousel not running
  FFh unsucessful
    AH = error code (01h,03h) (see Table 49-29)

**Details:** This function never returns if successful.

**INT 21h Function DF0Ah** *Software Carousel*

***GET MEMORY SIZE/PARTITION NUMBER OF CURRENT TASK***

**Registers at call:**
AX = DF0Ah

**Return Registers:**
AL = status
  00h Carousel not running
  01h successful
    BL = task number
    DX = memory size in KB
  FFh unsucessful
    AH = error code (01h,03h) (see Table 49-29)

---

**INT 21h Function DF0Bh** *Software Carousel*

***SET TASK SWITCH CALLBACK***

**Registers at call:**
AX = DF0Bh
BH = interrupt number or 00h
BL = function number to invoke on partition switch
CL = function number to call when it is safe for resident programs to perform DOS calls
DS:DX -> FAR function to call if BH=00h

**Return Registers:**
AL = status
  00h Carousel not running
  01h successful
  FFh unsucessful
    AH = error code (01h,03h) (see Table 49-29)

**Details:** The specified interrupt or FAR function is called with AH set to the appropriate one of the values specified in BL and CL, and BL set to the new task number.

The function specified by CL will not be called until the notification is enabled with Function DF0Ch.

**See also:** Function DF0Ch

---

**INT 21h Function DF0Ch** *Software Carousel*

***ENABLE DOS-CALL SAFETY NOTIFICATION***

**Registers at call:**
AX = DF0Ch

**Return Registers:**
AL = status
  00h Carousel not running
  01h successful

**See also:** Function DF0Bh

## SWELL (Swapping Shell)

SWELL.EXE is a TSR which swaps programs to disk when they EXEC a child process with INT 21h Function 4Bh.

All of the SWELL functions listed in this section conflict with the LaserPort interface (Chapter 15) and the Intel Image Processing Interface (Chapter 55), and potentially conflict with numerous other programs listed in Table 2-2.

---

**INT 2Fh Function CD00h** *SWELL.EXE*

***INSTALLATION CHECK***

**Registers at call:**
AX = CD00h

**Return Registers:**
AX = 00FFh installed
  BH = major version
  BL = minor version

---

**INT 2Fh Functions CD01h to CD03h** *SWELL.EXE*

***ENABLE/DISABLE SWELL***

**Registers at call:**
AX = CD01h suspend once

AX = CD02h suspend
AX = CD03h activate

**Return Registers:**
AX = 0000h

---

### INT 2Fh Functions CD04h and CD05h
### *TURN VERBOSE MODE OFF OR ON*

*SWELL.EXE*

**Registers at call:**
AX = CD04h turn off
AX = CD05h turn on

**Return Registers:**
AX = 0000h

### INT 2Fh Function CD06h
### *UNINSTALL*

*SWELL.EXE*

**Registers at call:**
AX = CD06h

**Return Registers:**
AX = status
0000h uninstalled
8002h programs still swapped, not uninstalled

### INT 2Fh Function CD07h
### *GET INFO*

*SWELL.EXE*

**Registers at call:**
AX = CD07h
ES:BX -> 32-byte buffer for info

**Return Registers:**
AX = status
0000h successful
ES:BX buffer filled
8001h buffer wrong size

**49-30 Format of info buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD 20h | (total size of buffer) |
| 02h | BYTE | suspend-once mode active if nonzero |
| 03h | BYTE | 00h active, 01h suspended |
| 04h | BYTE | 00h quiet, 01h verbose |
| 05h | BYTE | "Borland support" (allowing INT 21h Function 4B01h) on if nonzero |
| 06h | 26 BYTEs | *unused* |

### INT 2Fh Function CD08h
### *UNUSED*

*SWELL.EXE*

**Registers at call:**
AX = CD08h

**Return Registers:**
AX = FFFFh (error)

### INT 2Fh Functions CD09h and CD0Ah
### *TURN "BORLAND SUPPORT" OFF OR ON*

*SWELL.EXE*

**Registers at call:**
AX = CD09h turn off
AX = CD0Ah turn on

**Return Registers:**
AX = 0000h

## Switch-It

Switch-It is a task switcher supporting up to 100 programs simultaneously by Better Software Technology, Inc.

### INT 33h Function FFE6h
### *GET* UNKNOWN *PROGRAM*

*Switch-It v3.23*

**Registers at call:**
AX = FFE6h
CX = length of buffer
ES:DI -> buffer for program name

**Return Registers:**
ES:DI buffer filled

---

### INT 33h Function FFE7h *Switch-It v3.23*

***GET** UNKNOWN VALUE*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFE7h | AX = *unknown* |

---

### INT 33h Function FFE8h *Switch-It v3.23*

*UNDETERMINED FUNCTION*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFE8h | none |
| CX = length of name including terminating NUL | |
| DS:SI -> ASCIZ program pathname | |

---

### INT 33h Functions FFE9h to FFECh *Switch-It v3.23*

***SET** UNKNOWN VALUES*

| Registers at call: | Return Registers: |
|---|---|
| AX = function | none |

FFE9h set *unknown value*
  BX = *unknown*
FFEAh set *unknown value*
  BL = *unknown*
FFEBh set *unknown flag*
FFECh set *unknown value*
  BL = *unknown*

---

### INT 33h Functions FFEDh to FFEFh *Switch-It v3.23*

***GET** UNKNOWN VALUES*

| Registers at call: | Return Registers: |
|---|---|
| AX = function | |
| FFEDh | AX = *unknown* (0001h)<br>BX = *unknown* |
| FFEEh | AX = *unknown* |
| FFEFh | BX:AX -> *unknown* |

---

### INT 33h Function FFF0h *Switch-It v3.23*

***SET** UNKNOWN VALUE*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFF0h | none |
| BL = *unknown* | |

---

### INT 33h Function FFF1h *Switch-It v3.23*

***GET CONFIGURATION FILE***

| Registers at call: | Return Registers: |
|---|---|
| AX = FFF1h | BX:AX -> ASCIZ pathname of configuration file |

---

### INT 33h Function FFF2h *Switch-It v3.23*

***SET** UNKNOWN **FLAG***

| Registers at call: | Return Registers: |
|---|---|
| AX = FFF2h | AL = 01h |

---

### INT 33h Function FFF3h *Switch-It v3.23*

***GET** UNKNOWN VALUE*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFF3h | AX = *unknown* |

## INT 33h Function FFF4h

*Switch-It v3.23*

***SET** UNKNOWN VALUES*

**Registers at call:**
AX = FFF4h
BX = *unknown*
CX = *unknown*

**Return Registers:**
none

---

## INT 33h Function FFF5h and FFF6h

*Switch-It v3.23*

***GET** UNKNOWN VALUES*

**Registers at call:**
AX = FFF5h and FFF6h

**Return Registers:**
AX = *unknown*

---

## INT 33h Function FFF7h

*Switch-It v3.23*

***GET** UNKNOWN VALUE*

**Registers at call:**
AX = FFF7h
BX = index of *unknown*

**Return Registers:**
AX = *unknown*

---

## INT 33h Function FFF8h

*Switch-It v3.23*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = FFF8h
BX = *unknown*
CX = length of program name, including terminating NUL
DS:SI -> ASCIZ program pathname

**Return Registers:**
*unknown*

---

## INT 33h Function FFF9h

*Switch-It v3.23*

***NULL FUNCTION***

**Registers at call:**
AX = FFF9h

**Return Registers:**
none

---

## INT 33h Function FFFAh

*Switch-It v3.23*

***SET** UNKNOWN INDEX*

**Registers at call:**
AX = FFFAh
BX = index of program

**Return Registers:**
none

**See also:** Functions FFFBh and FFFCh

---

## INT 33h Function FFFBh

*Switch-It v3.23*

***GET** UNKNOWN VALUE*

**Registers at call:**
AX = FFFBh
BX = index of program

**Return Registers:**
AX = *unknown* (0000h or 0001h)

**See also:** Functions FFFAh and FFFCh

---

## INT 33h Function FFFCh

*Switch-It v3.23*

***CLEAR** UNKNOWN ITEM*

**Registers at call:**
AX = FFFCh
BX = index of program

**Return Registers:**
none

**See also:** Functions FFFAh and FFFCh

---

### INT 33h Function FFFDh

*Switch-It v3.23*

*GET MEMORY ADDRESSES*

**Registers at call:**
AX = FFFDh

**Return Registers:**
AX = *first available segment*
BX = paragraph of top of conventional memory
DX = PSP segment of SI.EXE

---

### INT 33h Function FFFEh

*Switch-It v3.23*

***INSTALLATION CHECK***

**Registers at call:**
AX = FFFEh

**Return Registers:**
DX = 5349h ("SI") if installed
BX = *unknown*

---

### INT 33h Function FFFFh

*Switch-It v3.23*

*UNDETERMINED FUNCTION*

**Registers at call:**
AX = FFFFh
BX = *unknown*

**Return Registers:**
none

## Miscellaneous

This section covers multitaskers and task switchers whose API is not extensive enough to warrant separate sections.

---

### INT 20h

*Minix*

***SEND/RECEIVE MESSAGE***

**Purpose:** Minix is a Version 7 Unix-compatible operating system by Andrew Tanenbaum.

**Registers at call:**
AX = process ID of other process
BX -> message
CX = operation (1 send, 2 receive, 3 send&receive)

**Return Registers:**
none

**Details:** The message contains the system call number (numbered as in V7 Unix(tm)) and the call parameters.

**Conflicts:** MS-DOS (Chapter 24), COMTROL HOSTESS (Chapter 57)

---

### INT 21h Function 2D01h, Subfunction 7820h

*PC-Mix*

***INSTALLATION CHECK***

**Registers at call:**
AX = 2D01h
CX = 7820h ('X ')
DX = 6D69h ('MI')

**Return Registers:**
AL = 00h if installed

---

### INT 21h Function 3000h, Subfunction 1234h

*CTask 2.0+*

***INSTALLATION CHECK***

**Registers at call:**
AX = 3000h
BX = 1234h
DS:DX -> 8-byte version string (DX < FFF0h)
"CTask21",00h for v2.1-2.2

**Return Registers:**
AL = DOS major version
AH = DOS minor version
CX:BX -> Ctask global data block

**Program:** CTask is a multitasking kernel for C written by Thomas Wagner.

**Details:** If the first eight bytes of the returned data block equal the eight bytes passed in, CTask is resident.

**Conflicts:** MS-DOS (Chapter 24)

**INT 7Fh Function 09h** *MultiLink Advanced*

### *SET TASK PRIORITY*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 09h | none |
| AL = priority (0-7) | |

**Details:** The installation check consists of ensuring that the interrupt vector is not pointing at segment 0000h, then checking whether the byte at offset 0000h in the interrupt handler's segment is E9h

**Conflicts:** see Table 2-4

## CHAPTER 50

# *Alternate Multiplex Interrupt*

The Multiplex Interrupt originally defined by Microsoft for use by its PRINT.COM and subsequently expanded in a series of contradictory specifications has become rather crowded with users in recent years (see Table 2-2), which prompted co-author Ralf Brown to propose an Alternate Multiplex Interrupt Specification (AMIS) in late 1991. A number of shareware authors have subsequently adopted the proposal, which is described in this chapter.

To be considered fully compliant with version 3.5 of the specification (described here), programs must implement at least Functions 00h, 02h (no resident uninstall code required), and 04h (return value 04h) described below. TSRs that provide hotkeys with which the user can activate them must also implement Function 05h. The absolute minimum fully-compliant implementation has an overhead of 64 bytes (80 bytes with Function 05h) plus 22 bytes per hooked interrupt (for the interrupt sharing protocol header and hook list entry, shown in Tables 50-1 and 50-6). A sample implementation including example TSRs and utility programs may be found in the AMISLIB package, distributed as AMISLnnn.ZIP (AMISL091.ZIP as of this writing).

Users of the alternate multiplex interrupt should adhere to the IBM interrupt sharing protocol (see Table 50-1), which will permit removal of TSRs in arbitrary order and interrupt handler reordering. All AMIS-compliant TSRs should be removable, though they need not keep the code for removing themselves resident; it is acceptable for a separate program to perform the removal.

AMIS programs should not use fixed multiplex numbers; rather, a program should scan all multiplex numbers from 00h to FFh, remembering the first unused multiplex in case the program is not yet installed. For multiplex numbers which are in use, the program should compare the first 16 bytes of the signature string to determine whether it is already installed on that multiplex number. If not previously installed, it should use the first free multiplex number.

Functions other than 00h are not valid unless a program is installed on the selected multiplex number. Please notify Ralf Brown if you choose to follow this proposal. The signature and a list of the private API calls you use would be appreciated, as well.

**50-1 Format of interrupt sharing protocol interrupt handler entry point:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | short jump to actual start of interrupt handler, immediately following this data block (EBh 10h) |
| 02h | DWORD | address of next handler in chain |
| 06h | WORD | signature 424Bh |
| 08h | BYTE | EOI flag<br>00h software interrupt or secondary hardware interrupt handler<br>80h primary hardware interrupt handler (will issue EOI) |
| 09h | 2 BYTEs | short jump to hardware reset routine<br>must point at a valid FAR procedure (may be just RETF) |
| 0Bh | 7 BYTEs | reserved (0) |

## INT 2Dh Function 00h

*AMIS v3.5.1*

### *INSTALLATION CHECK*

**Registers at call:**
AL = 00h
AH = multiplex number

**Return Registers:**
AL = status
00h if free
FFh if multiplex number in use
CX = binary version number (CH = major, CL = minor)
DX:DI -> signature string (see Table 50-2) identifying the program using the multiplex number (see Table 50-3)

**Details:** The signature string and description may be used by memory mappers to display the installed programs.

**50-2 Format of signature string:**

| *Offset* | *Size* | | *Description* |
|---|---|---|---|
| 00h | 8 | BYTEs | blank-padded manufacturer's name (possibly abbreviated) |
| 08h | 8 | BYTEs | blank-padded product name |
| 10h | 64 | BYTEs | ASCIZ product description (optional, may be a single 00h) |

**Note:** It is not necessary to reserve a full 64 bytes for the description, just enough to store the actual ASCIZ string.

**50-3 AMIS signatures known to be in use:**

| *Manufacturer* | *Product* | *Description* |
|---|---|---|
| 'Byrial J' | 'EKLAVO ' | permits keyboard entry of Esperanto accented letters |
| 'CoveSoft' | 'Burnout+' | shareware screen saver Burnout Plus |
| 'Crynwr ' | 'SPELLER ' | TSR spelling-checker |
| 'CSJewell' | 'Modula3L' | Curtis Jewell's Modula-3 compiler (non-TSR) |
| 'GraySoft' | 'GIPC    ' | GraySoft's Inter-Process Communications driver |
| 'heathh ' | 'Monitor ' | |
| 'J. Berry' | 'RATSR ' | RemoteAccess Network Manager workstation module |
| 'JWB ' | 'RAMLIGHT' | James Birdsall's on-screen RAMdisk activity indicator |
| 'Nildram ' | 'ST ' | Screen Thief graphics screen grabber |
| 'R-Ware ' | 'dLite' | run-time data decompression TSR |
| 'Ralf B ' | 'FASTMOUS' | example TSR included with sample AMIS library code |
| 'Ralf B ' | 'NOLPT n ' | example TSR—turn LPTn into bit-bucket |
| 'Ralf B ' | 'NOTE    ' | example TSR—popup note-taker |
| 'Ralf B ' | 'RBkeyswp' | RBkeyswap v3.0+—swap Esc/ and LCtrl/CapsLock keys |
| 'Ralf B' | 'SWITCHAR' | example TSR—add switchar() support removed from DOS5 |
| 'Ralf B ' | 'VGABLANK' | example TSR—VGA-only screen blanker |
| 'Sally IS' | 'Mdisk    ' | removeable, resizeable RAMdisk |
| 'Sally IS' | 'Scr2Tex ' | screen dumper with output in (La)Tex format |
| 'Thaco ' | 'NEST' | Eirik Pedersen's programmer's delimiter matcher |
| 'TifaWARE' | 'EATMEM' | George A. Theall's public domain memory restrictor for testing programs (v1.1+) |
| 'TifaWARE' | 'RECALL ' | public domain commandline editor and history (v1.2+) |
| 'Todd    ' | 'XPTR2' | PC-to-Transputer interface by Todd Radel |

## INT 2Dh Function 01h

*AMIS v3.5.1*

### *GET ENTRY POINT*

**Registers at call:**
AL = 01h
AH = multiplex number

**Return Registers:**
AL = 00h if all API calls via INT 2Dh
AL = FFh if entry point supported
DX:BX -> entry point for bypassing interrupt chain

---

### INT 2Dh Function 02h

*AMIS v3.5.1*

***UNINSTALL***

**Registers at call:**

AL = 02h
AH = multiplex number
DX:BX = return address for successful uninstall (may be ignored by TSR)

**Return Registers:**

AL = status
- 00h not implemented
- 01h unsuccessful
- 02h can not uninstall yet, will do so when able
- 03h safe to remove, but no resident uninstaller (TSR still enabled)
  - BX = segment of memory block with resident code
- 04h safe to remove, but no resident uninstaller (TSR now disabled)
  - BX = segment of memory block with resident code
- 05h not safe to remove now, try again later
- FFh successful

return at DX:BX with AX destroyed if uninstall is successful and the TSR honors the specific return address

---

### INT 2Dh Function 03h

*AMIS v3.5.1*

***REQUEST POP-UP***

**Registers at call:**

AL = 03h
AH = multiplex number

**Return Registers:**

AL = status
- 00h not implemented or TSR is not a pop-up
- 01h can not pop up at this time, try again later
- 02h can not pop up yet, will do so when able
- 03h already popped up
- 04h unable to pop up, user intervention required
  - BX = standard reason code (see Table 50-4)
  - CX = application's reason code if nonzero
- FFh TSR popped up and was exited by user
  - BX = return value (see Table 50-5)

**50-4 Standard reason codes:**

| | |
|---|---|
| 0000h | unknown failure |
| 0001h | interrupt chain passes through memory which must be swapped out to pop up |
| 0002h | swap-in failed |

**50-5 TSR return values:**

| | |
|---|---|
| 0000h | no return value |
| 0001h | TSR unloaded |
| 0002h-00FFh | reserved |
| 0100h-FFFFh | application-dependent |

---

### INT 2Dh Function 04h

*AMIS v3.5.1*

***DETERMINE CHAINED INTERRUPTS***

**Registers at call:**

AL = 04h
AH = multiplex number
BL = interrupt number (except 2Dh)

**Return Registers:**

AL = status
- 00h not implemented
- 01h (obsolete) unable to determine
- 02h (obsolete) interrupt hooked
- 03h (obsolete) interrupt hooked, address returned
  - DX:BX -> TSR's interrupt BL handler
- 04h list of hooked interrupts returned
  - DX:BX -> interrupt hook list (see Table 50-6)
- FFh interrupt not hooked

**Details:** Since INT 2Dh is known to be hooked, the resident code need not test for BL=2Dh (to minimize its size), and the return value is therefore undefined in that case.

BL is ignored if the TSR returns AL=04h; in that case, the caller needs to scan the return list rather than making additional calls to this function. If the return is not 00h or 04h, then the caller must cycle through the remaining interrupt numbers it wishes to check.

Return values 01h thru 03h are disparaged and will be removed from the next version of this specification; they are included for compatibility with version 3.3, though they were probably never used in any implementation.

### 50-6 Format of interrupt hook list [array]:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | interrupt number (last entry in array is 2Dh) |
| 01h | WORD | offset within hook list's segment of the interrupt handler<br>this will point at the initial short jump of the interrupt sharing protocol header (see Table 50-1) |

---

## INT 2Dh Function 05h

*AMIS v3.5.1*

### *GET HOTKEYS*

**Registers at call:**
AL = 05h
AH = multiplex number

**Return Registers:**
AL = status
00h not implemented
FFh supported
DX:BX -> hotkey list (see Table 50-7)

### 50-7 Format of hotkey list:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | type of hotkey checking<br>bit 0: checks before chaining INT 09h<br>bit 1: checks after chaining INT 09h<br>bit 2: checks before chaining INT 15h Function 4Fh<br>bit 3: checks after chaining INT 15h Function 4Fh<br>bit 4: checks on INT 16h Function 00h,01h,02h<br>bit 5: checks on INT 16h Function 10h,11h,12h<br>bit 6: checks on INT 16h Function 20h,21h,22h<br>bit 7: reserved (0) |
| 01h | BYTE | number of hotkeys (may be zero if TSR can disable hotkeys) |
| 02h | 6N BYTEs | array of hotkey definitions (see Table 50-8) (one per hotkey, first should be primary hotkey) |

### 50-8 Format of hotkey definition:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | hotkey scan code (00h/80h if shift states only)<br>hotkey triggers on release if bit 7 set |
| 01h | WORD | required shift states (see Table 50-9) |
| 03h | WORD | disallowed shift states (see Table 50-9) |
| 05h | BYTE | flags<br>bit 0: hotkey chained before processing<br>bit 1: hotkey chained after processing<br>bit 2: others should pass through this hotkey so that it can be monitored<br>bit 3: hotkey will not activate if other keys pressed/released before hotkey press is completed<br>bit 4: this key is remapped into some other key<br>bits 5-7: reserved (0) |

**Notes:** Except for bit 7, the shift states correspond exactly to the return values from INT 16h Function 12h. A set bit in the required states word indicates that the corresponding shift state must be active when the hotkey's scan code is received for the hotkey to be recognized; a clear bit means that the corresponding state may be ignored. A set bit in the disallowed shift states word indicates that the corresponding shift state must be inactive.

If bit 2 is set, either control key may be pressed for the hotkey; if bits 8 and 10 are both set, then both control keys must be pressed. Similarly for bits 3 and 9/11, as well as 7 and 0/1.

For the disallowed-states word, if one of the "either" bits is set, then both the corresponding left bit and right bit must be set.

Examples:

Ctrl-Alt-Del monitoring: 53h 000Ch 0003h 06h
Alt-key tap (DESQview): B8h 0000h 0007h 08h
Shf-Shf-N (NOTE.COM): 31h 0003h 000Ch 00h

### 50-9 Bitfields for shift states:

| | |
|---|---|
| bit 0 | right shift pressed |
| bit 1 | left shift pressed |
| bit 2 | either control key pressed |
| bit 3 | either Alt key pressed |
| bit 4 | ScrollLock active |
| bit 5 | NumLock active |
| bit 6 | CapsLock active |
| bit 7 | either shift key pressed |
| bit 8 | left control key pressed |
| bit 9 | left Alt key pressed |
| bit 10 | right control key pressed |
| bit 11 | right Alt key pressed |
| bit 12 | ScrollLock pressed |
| bit 13 | NumLock pressed |
| bit 14 | CapsLock pressed |
| bit 15 | SysRq key pressed |

## INT 2Dh Functions 06h to 0Fh

*AMIS v3.5.1*

### *RESERVED FUNCTIONS*

**Registers at call:**
AH = multiplex number
AL = function
06h-0Fh reserved for future enhancements

**Return Registers:**
AL = 00h (not implemented)

**See also:** INT 2Fh

## INT 2Dh Functions 10h to FFh

*AMIS v3.5.1*

### *APPLICATION-SPECIFIC FUNCTIONS*

**Registers at call:**
AL = 10h to FFh
AH = multiplex number
other registers as appropriate for program

**Return Registers:**
as appropriate for program

## CHAPTER 51

# *TSR Libraries*

TSR programming being somewhat of a "mystic art", particularly where DOS access is concerned, it is not surprising that a number of libraries have been created to simplify creation of resident programs. These are listed in this chapter.

---

### INT 15h

*Microsoft TSR Specification*

***API***

No additional information was available at the time of writing.

---

### INT 16h Function 5453h

*TSRBONES*

***INSTALLATION CHECK***

**Program:** TSRBONES is a skeletal TSR framework by Robert Curtis Davis.

| Registers at call: | Return Registers: |
|---|---|
| AX = 5453h ('TS') | AX = 4553h if installed |
| BX = 5242h ('RB') | BX = 4F4Eh if installed |
| CX = 4F4Eh ('ON') | CX = 5242h if installed |
| DX = 4553h ('ES') | DX = 5453h if installed |

**Details:** These values are the default as the TSRBONES package is distributed, but will normally be changed when implementing an actual TSR with the TSRBONES skeleton.

**See also:** AMIS INT 2Dh (Chapter 50)

---

### INT 16h Function 55FFh

*Swap Utilities*

*UNDETERMINED FUNCTION*

| Registers at call: | Return Registers: |
|---|---|
| AX = 55FFh | none |
| BX >= 0004h | |
| CX = function<br>0000h set *unknown* flag<br>other clear *unknown* flag | |

**Details:** Present in SWAPSH and SWAPDT v1.77j, distributed with PC Tools v7, as well as the Trusted Access SCRNBLNK.COM; this may be part of the standard TesSeRact library.

**See also:** TesSeRact INT 2Fh Function 5453h below

---

### INT 16h Function E0E0h

*TurboPower TSRs*

***ALTERNATE INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = E0E0h | AX = 1F1Fh if installed<br>DWORD 0040h:00F0h -> last data block in TSR list (see Table 51-1) |

**Details:** The returned TSR list provides support for communication among TSRs built with TurboPower's Turbo Professional and Object Professional libraries for Turbo Pascal.

**See also:** Function F0F0h

## INT 16h Function F0F0h
## INSTALLATION CHECK

*TurboPower TSRs*

**Registers at call:**
AX = F0F0h

**Return Registers:**
AX = 0F0Fh if installed
ES:DI -> last data block in TSR list (see Table 51-1)

**Details:** The returned TSR list provides support for communication among TSRs built with TurboPower's Turbo Professional and Object Professional libraries for Turbo Pascal.
**Conflicts:** Compaq 386 (Chapter 7)
**See also:** Function E0E0h

### 51-1 Format of data block:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to program tag (counted ASCII string) |
| 04h | WORD | interface version number (0400h) |
| 06h | DWORD | pointer to command entry point |
| 0Ah | DWORD | pointer to previous data block (0000h:0000h if none) |
| 0Eh | DWORD | pointer to next data block (0000h:0000h if none) |
| **---swappable TSRs only---** | | |
| 12h | DWORD | pointer to swapping data |
| 16h | DWORD | pointer to user data |
| | *additional fields, if any, unknown* | |

## INT 2Fh
## INSTALLATION CHECK

*BMB Compuscience Canada Utilities Interface*

**Registers at call:**
AH = dynamically-assigned multiplex number (based upon a search for one which doesn't answer installed)
AL = 00h installation check
ES:DI = EBEBh:BEBEh

**Return Registers:**
AL = status
00h not installed
01h not installed, not OK to install
FFh installed; if ES:DI was EBEBh:BEBEh on entry, ES:DI will point to a signature string

**Details:** The signature string is of the form 'MMMMPPPPPPPPvNNNN', where MMMM is a short form of the manufacturer's name, PPPPPPPP is a product name and NNNN is the product's version number.
**Conflicts:** see Table 2-2

## INT 2Fh
## API

*Ross Wentworth's Turbo Pascal POPUP LIBRARY*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = programmer-selected multiplex number | |
| AL = function | |
| 00h installation check | AL = FFh if installed |
| 01h get TSR interrupt vectors | DX:AX -> vector table (see Table 51-2) |
| 02h get TSR code segment | AX = code segment for all interrupt handlers |
| 03h call user exit routine and release TSR's memory | |
| 04h get signature string | DX:AX -> counted string containing signature |

**Registers at call:**
AH = programmer-selected multiplex number
AL = function
05h get TSR's INT 2Fh handler

06h enable/disable TSR
  BL = new state (00h disabled, 01h enabled)
07h activate TSR (popup if not disabled)

08h get hotkeys
  BL = which hotkey (00h = hotkey 1, 01h = hotkey 2)
09h set hotkey
  BL = which hotkey (00h = hotkey 1, 01h = hotkey 2)
  CX = new hotkey (CH = keyflags, CL = scancode)
0Ah-1Fh reserved

**Return Registers:**

DX:AX -> INT 2Fh handler

AX = hotkey (AH = keyflags,
  AL = scancode)

**Conflicts:** see Table 2-2

**51-2 Format of vector table entry:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | vector number (00h = end of table) |
| 01h | DWORD | original vector |
| 05h | WORD | offset of interrupt handler in TSR's code segment |

---

**INT 2Fh** *CiriSOFT Spanish University of Valladolid TSR's Interface*

***API***

**Registers at call:**
AH = dynamically-assigned multiplex number (based upon a search for a multiplex number from C0h to FFh which doesn't answer installed)
AL = 00h installation check
ES:DI = 1492h:1992h

**Return Registers:**
AL = status
  00h not installed
  01h not installed, not OK to install
  FFh installed; if ES:DI was 1492h:1992h on entry, ES:DI will point to author_name_ver table (see Table 51-3)
AH = FFh

**Details:** This interface permits advanced communication with TSRs: it is possible to make a generic uninstall utility, advanced TSR relocator programs in order to fit fragmented memory areas, etc. See also: AMIS INT 2Dh,Compuscience INT 2Fh

**Conflicts:** see Table 2-2

**51-3 Format of author_name_ver table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -16 | WORD | segment of the start of the resident TSR code (CS in programs with PSP, XMS upper memory segment if installed as UMB...) |
| -14 | WORD | offset of the start of the resident TSR code (frequently 100h in *.COM programs and 0 in upper memory TSR's). |
| -12 | WORD | memory used by TSR (in paragraphs). Knowing the memory area used by TSR is possible to determine if hooked vectors are still pointing it (and if it is safe to uninstall). |
| -10 | BYTE | characteristics byte<br>bits 0-2: program type<br>000 normal program (with PSP)<br>001 upper XMS memory block (needed HIMEM.SYS function to free memory when uninstalling)<br>010 device driver (*.SYS)<br>011 device driver in EXE format<br>1xx others (reserved)<br>bits 3-6 reserved<br>bit 7 set if extra_table defined and supported |
| -9 | BYTE | number of multiplex entry used (redefinition available). Note that the TSR must be use THIS variable in it's INT 2Fh handler. |

| | | |
|---|---|---|
| -8 | WORD | offset to vector_area table (see Table 51-4) |
| -6 | WORD | offset to extra_area table (see bit 7 in offset -10 and below) |
| -4 | 4 BYTEs | signature string "*##*" |
| 00h | var | "AUTHOR:PROGRAM_NAME:VERSION",0 (variable length, this area is used in order to determine if the TSR is already resident and it's version code; the ':' char is used as delimiter) |

**51-4 Format of vector_area table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -1 | BYTE | number of vectors intercepted by TSR |
| 00h | BYTE | first vector number |
| 01h | DWORD | first vector pointer before installing the TSR |
| 05h | BYTE | second vector number |
| 06h | DWORD | second vector pointer before installing the TSR |
| 0Ah | ... | (and so on) |

**Note:** The TSR must use these variables to invoke the previous interrupt handler routines.

**51-5 Format of extra_area table (needed only to improve relocation feature):**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | offset to external_ctrl table (0 if not supported) |
| 02h | WORD | reserved for future use (0) |

**51-6 Format of external_ctrl table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | bit 0: TSR is relocatable (no absolute segment references) |
| 01h | WORD | offset to a variable which can activate/inhibit the TSR |
| ---And if bit 0 in offset 00h is off: | | |
| 03h | DWORD | pointer to ASCIZ pathname for executable file which supports /SR parameter (silent installation & inhibit) |
| 07h | DWORD | pointer to first variable to initialize on the copy reloaded from the previous TSR still resident |
| 0Bh | DWORD | pointer to last variable (all variables packed in one block) |

## INT 2Fh Function 3900h
### *COMPATIBILITY MODE*

*Kingswood TSR INTERFACE*

**Registers at call:**
AX = 3900h

**Return Registers:**
AL = status
00h not installed
FFh one or more TSRs are using this interface

**Details:** This function is provided to that the multiplex number will appear used to other programs.
**See also:** Function 39h Subfunction 00h

## INT 2Fh Function 39h, Subfunction 00h
### *INSTALLATION CHECK*

*Kingswood TSR INTERFACE*

**Registers at call:**
AH = 39h
BL = 00h
AL = TSR ID number (01h-FFh, currently only 01h-0Eh used) (see Table 51-7)

**Return Registers:**
AL = status
00h not installed
FFh installed
DX = segment address of resident module (see Table 51-8)

**Details:** All of Kingswood Software's TSRs use this interface. Usually the resident module is installed by allocating a block of upper memory, setting its owner ID to 0008h (DOS data), and filling the MCB name field with the TSR's name.
**See also:** Function 3900h, Function 39h Subfunction 01h

### 51-7 Values for TSR ID number:

| | |
|---|---|
| 01h | TSR Windows |
| 02h | NOBUSY |
| 03h | CD STACK |
| 04h | DISK WATCH |
| 05h | PUSHBP |
| 06h | ALIAS |
| 07h | KEYMACRO |
| 08h | SLOWDOWN |
| 09h | ANSIGRAB |
| 0Ah | TEE |
| 0Bh | FASTMOUS |
| 0Ch | EXTWILD |
| 0Dh | BREAKOUT |
| 0Eh | STOPDISK |

### 51-8 Format of TSR modules:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 4 BYTEs | signature "FTSR" |
| 04h | WORD | segment address of this module |
| 06h | WORD | number of words to skip (usually 0000h if no PSP present) |
| 08h | N WORDs | module-defined data that must be at a fixed segment offset (usually only a PSP if file access is required) |
| | 5N BYTEs | interrupt list (see Table 51-9) |
| | BYTE | FFh terminator |

### 51-9 Format of interrupt list entry:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | interrupt number (00h-FEh) |
| 01h | WORD | offset within segment of DWORD pointer to previous interrupt |
| 03h | WORD | offset within segment of begin of interrupt handler code |

---

## INT 2Fh Function 39h, Subfunction 01h

*Kingswood TSR INTERFACE*

### *REMOVAL CHECK*

**Registers at call:**

AH = 39h
BL = 01h
AL = TSR ID number (01h-FFh)
(see Table 51-7)

**Return Registers:**

AL = status
00h not ready to be removed
FFh resident module may be removed by deassigning the interrupts hooked by the TSR and freeing its memory block
AH,BX,CX,DX,ES may be destroyed

**See also:** Function 3900h, Function 39h Subfunction 00h

---

## INT 2Fh Function 39h

*Kingswood TSR INTERFACE*

### *APPLICATION-SPECIFIC FUNCTION CALLS*

**Registers at call:**

AH = 39h
BL = function number (02h-FFh)
AL = TSR ID number (01h-FFh)
CX,DX,SI,DI,DS,ES may contain parameters

**Return Registers:**

as appropriate for the called function

**See also:** Function 3900h, Function 39h Subfunction 00h, Function 3901h Subfunction 02h (Chapter 66)

### INT 2Fh Function 5453h

*TesSeRact RAM-RESIDENT PROGRAM INTERFACE*

### *INTERFACE*

**Registers at call:**
AX = 5453h
BX = subfunction
00h installation check
CX = 0000h
DS:SI -> 8-char blank-padded name

**Return Registers:**
AX = status
FFFFh installed
CX = ID number of already-installed copy
else, not installed
CX = ID number for TSR when installed

**Registers at call:**
01h get user parameters
CX = TSR ID number

**Return Registers:**
AX = status
0000h successful
ES:BX -> user parameter block (see Table 51-11)
nonzero failed

**Registers at call:**
02h check if hotkey in use
CL = scan code of hot key

**Return Registers:**
AX = status
FFFFh hot key conflicts with another TSR
otherwise safe to use the hotkey

**Registers at call:**
03h replace default critical error handler
CX = TSR ID number
DS:SI -> new routine for INT 24h

**Return Registers:**
AX = nonzero, unable to install new handler

**Registers at call:**
04h get internal data area
CX = TSR ID number

**Return Registers:**
AX = status
0000h successful
ES:BX -> TSR's internal data area (see Table 51-12)
nonzero if TSR not found

**Registers at call:**
05h set multiple hot keys
CX = TSR ID number
DL = number of additional hot keys to allocate
DS:SI -> table of hot keys
BYTE hotkey scan code
BYTE hotkey shift state
BYTE flag value to pass to TSR (nonzero)

**Return Registers:**
AX = nonzero if unable to install hot keys

**Registers at call:**
06h - 0Fh reserved
10h enable TSR
CX = TSR ID number

**Return Registers:**
AX = nonzero if unable to enable

**Registers at call:**
11h disable TSR
CX = TSR ID number

**Return Registers:**
AX = nonzero if unable to disable

**Registers at call:**
12h unload TSR
CX = TSR ID number

**Return Registers:**
AX = nonzero if invalid TSR number

**Details:** If any interrupts used by TSR have been grabbed by another TSR, the TesSeRact routines will wait until it is safe to remove the indicated TSR from memory.

**Registers at call:**
13h restart TSR
CX = TSR ID number of TSR which was unloaded but is still in memory

**Return Registers:**
AX = nonzero if unable to restart TSR

**Registers at call:**
14h get status word
CX = TSR ID number

**Return Registers:**
AX = status
FFFFh invalid ID number
else successful
BX = bit flags

**Registers at call:**
15h set status word
CX = TSR ID number
DX = new bit flags

**Return Registers:**
AX = nonzero if unable to set status word

**Registers at call:**
16h get INDOS state at popup
CX = TSR ID number

**Return Registers:**
AX = 0000h successful
BX = value of INDOS flag

**Registers at call:**
17h - 1Fh reserved
20h call user procedure
CX = TSR ID number
ES:DI -> user-defined data

**Return Registers:**

AX = 0000h successful

**Registers at call:**
21h stuff keystrokes into keyboard buffer
CX = TSR ID number
DL = speed
00h stuff keystrokes only when buffer is empty
01h stuff up to four keystrokes per clock tick
02h stuff up to 15 keystrokes per clock tick
DH = scan code flag
if zero, buffer contains alternating ASCII and scan codes
if nonzero, buffer contains only ASCII codes
SI = number of keystrokes
ES:DI -> buffer to stuff

**Return Registers:**
AX = status
0000h success
F0F0h user aborted with ^C or ^Break
else unable to stuff keystrokes

**Registers at call:**
22h (v1.10) trigger popup
CX = TSR ID number

**Return Registers:**
AX = status
0000h success, TSR will either pop up or beep to indicate that it is unable to pop up
nonzero invalid ID number

**Registers at call:**
23h (v1.10) invoke TSR's background function
CX = TSR ID number

**Return Registers:**
AX = status
0000h success
FFFFh not safe to call background function
else invalid ID number

**Registers at call:**
24h - 2Fh reserved

**Return Registers:**
none

**Details:** Borland's THELP.COM popup help system for Turbo Pascal and Turbo C (versions 1.x and 2.x only) fully supports the TesSeRact API, as do the SWAP?? programs by Innovative Data Concepts.

AVATAR.SYS supports functions 00h and 01h (only the first three fields of the user parameter block) using the name "AVATAR ".

**Conflicts:** POWER.EXE (Chapter 19)

**See also:** INT 16h Function 55FFh, AMIS INT 2Dh (Chapter 50), THELP Function CAFEh (Chapter 66)

### 51-10 Values for TesSeract names:

| | |
|---|---|
| "AVATAR " | AVATAR.SYS |
| "SCRNBLNK" | Trusted Access screen blanker |

### 51-11 Format of User Parameter Block:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 8 BYTEs | blank-padded TSR name |
| 08h | WORD | TSR ID number |
| 0Ah | DWORD | bitmap of supported functions |
| 0Eh | BYTE | scan code of primary hotkey<br>00h = pop up when shift states match<br>FFh = no popup (if shift state also FFh) |
| 0Fh | BYTE | shift state of primary hotkey<br>FFh = no popup (if scan code also FFh) |
| 10h | BYTE | number of secondary hotkeys |
| 11h | DWORD | pointer to extra hotkeys set by func 05h |
| 15h | WORD | current TSR status flags |
| 17h | WORD | PSP segment of TSR |
| 19h | DWORD | DTA for TSR |
| 1Dh | WORD | default DS for TSR |
| 1Fh | DWORD | stack at popup |
| 23h | DWORD | stack at background invocation |

### 51-12 Format of TSR internal data area:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | revision level of TesSeRact library |
| 01h | BYTE | type of popup in effect |
| 02h | BYTE | INT 08h occurred since last invocation |
| 03h | BYTE | INT 13h occurred since last invocation |
| 04h | BYTE | active interrupts |
| 05h | BYTE | active soft interrupts |
| 06h | BYTE | DOS major version |
| 07h | BYTE | how long to wait before popping up |
| 08h | DWORD | pointer to INDOS flag |
| 0CH | DWORD | pointer to DOS critical error flag |
| 10h | WORD | PSP segment of interrupted program |
| 12h | WORD | PSP segment of prog interrupted by INT 28h |
| 14h | DWORD | DTA of interrupted program |
| 18h | DWORD | DTA of program interrupted by INT 28h |
| 1Ch | WORD | SS of interrupted program |
| 1Eh | WORD | SP of interrupted program |
| 20h | WORD | SS of program interrupted by INT 28h |
| 22h | WORD | SP of program interrupted by INT 28h |
| 24h | DWORD | INT 24h of interrupted program |
| 28h | 3 WORDs | DOS 3+ extended error info |
| 2Eh | BYTE | old BREAK setting |
| 2Fh | BYTE | old VERIFY setting |
| 30h | BYTE | were running MS WORD 4.0 before popup |
| 31h | BYTE | MS WORD 4.0 special popup flag |
| 32h | BYTE | enhanced keyboard call in use |
| 33h | BYTE | delay for MS WORD 4.0 |

## INT 2Fh Function 5453h

11 times (for INTs 08h,09h,13h,16h,1Ch,21h,28h,2Fh,1Bh,23h, and 24h):

| | |
|---|---|
| DWORD | old interrupt vector |
| BYTE | interrupt number |
| WORD | offset in TesSeRact code segment of new interrupt handler |

---

## INT 2Fh Function DDh

*CappaCom programs*

***API***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = DDh | |
| AL = 00h general installation check | AL = FFh if any CappaCom programs are resident |
| AL = FEh get info | ES:BX -> TSR info list (see Table 51-13) |
| **Registers at call:** | **Return Registers:** |
| AH = DDh | AL = status (bit 7 set on error) |
| AL = program identifier | AL = 81h if unknown function |
| BH = function | |
| FDh get version | BX = version |
| FFh installation check | AL = FFh if installed |
| | BX = version |
| | ES = segment of resident code |
| others vary by program | |

**Details:** CappaCom was originally SoftCom but changed its name due to a trademark conflict.

**Conflicts:** xDISK (Chapter 10)

**51-13 Format of TSR info list:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 9 BYTEs | blank-padded ASCIZ program name |
| 09h | BYTE | program ID |
| 0Ah | WORD | program's PSP segment |
| 0Ch | WORD | program version (major in high byte) |
| 0Eh | DWORD | pointer to next item in info list or 0000h:0000h |
| 12h | BYTE | number of interrupts hooked |
| 13h | 5 BYTEs | interrupt numbers hooked by program |
| 18h | 8 BYTEs | reserved |

---

## INT 66h Function FFFBh

*MicroHelp Stay-Res Plus*

*UNDETERMINED FUNCTION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = FFFBh | *unknown* |
| BX = FFFBh | |
| *others, if any, unknown* | |

**Conflicts:** see Table 2-3

**See also:** Function FFFEh, AMIS INT 2Dh (Chapter 50)

---

## INT 66h Function FFFEh

*MicroHelp Stay-Res/Stay-Res Plus*

***UNINSTALL***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = FFFEh | only returns if unsuccessful |
| BX = FFFEh | |

**Details:** This installation check is for the interrupt handler to begin with the bytes FBh 9Ch or 9Ch FAh, and the program name (not case-sensitive) to appear at offset 0005h (older versions) or the offset returned by Function FFFFh Subfunction FFF0h in the interrupt handler segment.

Programs which use Stay-Res include ThesPlus (program name "THESPLUS") and Personal Calendar (program name "CAL") by Paul Muñoz-Colman.
**Conflicts:** see Table 2-3
**See also:** Functions FFFBh and FFFFh, AMIS INT 2Dh (Chapter 50)

---

### INT 66h Function FFFFh
*MicroHelp Stay-Res Plus*

### *FIND PROGRAM NAME*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFFFh | DI = offset of program name in interrupt handler segment |
| BX = FFF0h | |

**Conflicts:** see Table 2-3
**See also:** Functions FFFBh and FFFEh, AMIS INT 2Dh (Chapter 50)

## CHAPTER 52

# *FastGraph/Light*

FGDRIVER is the external video driver for the shareware Fastgraph/Light by Ted Gruber Software. Although the interrupt calls are not documented directly, the high-level function bindings which map directly into interrupt calls are documented.

The FGDRIVER installation check consists of testing for the signature "FG" ten bytes beyond the start of the INT 62h interrupt handler. As with all programs using interrupts in the range 60h to 67h, there are numerous potential conflicts, which are listed in Table 2-3.

---

### INT 62h Function 0000h
### *"FG_GETMODE" - GET CURRENT VIDEO MODE NUMBER*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0000h
ES:BX -> *unknown*

**Return Registers:**
AX = current video mode number

**See also:** Functions 0001h and 0006h

---

### INT 62h Function 0001h
### *"FG_SETMODE" - SELECT VIDEO MODE AND INITIALIZE*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0001h
BX = new video mode or FFFFh for current mode
ES:DX -> *unknown*

**Return Registers:**
none

**Details:** Video modes are the same as the BIOS video modes except as shown in Table 52-1.

This call resets the active video page to page 0000h, the clipping region to the entire screen, text rows to 25, etc.

**See also:** Functions 0000h and 0002h, BIOS INT 10h Function 00h (Chapter 8)

.comment snake into two columns

**52-1 FastGraph/Light video modes:**

| | |
|---|---|
| 0Bh | Hercules graphics 720x348 |
| 0Ch | Hercules graphics 320x200 |
| 15h | VGA graphics 320x400x256 |
| 16h | VGA graphics 320x240x256 |
| 17h | VGA graphics 320x480x256 |

---

### INT 62h Function 0002h
### *"FG_TESTMODE" - CHECK IF VIDEO MODE AVAILABLE*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0002h
BX = desired video mode (00h-17h)

CX = required number of video pages
(ignore memory size if <= 0)

**Return Registers:**
AX = status
0000h mode not available with requested number of pages
0001h mode is available

**See also:** Functions 0001h, 0003h, 0004h, and 0005h

---

### INT 62h Function 0003h
### *"FG_BESTMODE" - GET BEST VIDEO MODE GIVEN RESOLUTION*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0003h
BX = horizontal resolution
CX = vertical resolution
DX = number of video pages required
(both physical and virtual)

**Return Registers:**
AX = proposed video mode number or FFFFh if no matching video mode

**See also:** Functions 0002h and 0004h

---

### INT 62h Function 0004h
### *"FG_AUTOMODE" - GET VIDEO MODE WITH MOST FEATURES*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0004h

**Return Registers:**
AX = proposed video mode number

**See also:** Functions 0002h, 0003h, and 0005h

---

### INT 62h Function 0005h
### *"FG_EGACHECK" - GET INFO ABOUT ACTIVE EGA DISPLAY*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0005h

**Return Registers:**
AX = number of 64K banks of video memory, or 0000h if no EGA or EGA without an Enhanced Color Display

**See also:** Functions 0002h and 0003h

---

### INT 62h Function 0006h
### *"FG_RESET" - ERASE SCREEN AND RESTORE SCREEN ATTRIBUTES*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0006h

**Return Registers:**
none

**Details:** This call is ignored in graphics modes. The screen attributes are only restored if ANSI.SYS is loaded.

**See also:** Function 0000h

---

### INT 62h Function 0007h
### *"FG_CURSOR" - SPECIFY WHETHER TEXT CURSOR IS VISIBLE*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0007h
BX = new state (0000h invisible, 0001h visible)

**Return Registers:**
none

**Details:** This call is ignored in text modes.

---

### INT 62h Functions 0008h and 0009h
### *UNUSED*

*FGDRIVER v1.10*

**Registers at call:**
AX = 0008h and 0009h

**Return Registers:**
AX = 0000h

**INT 62h Function 000Ah** *FGDRIVER v1.10*
***"FG_GETMAXX" - GET MAXIMUM COLUMN IN SCREEN SPACE*** *(partially doc.)*

**Registers at call:**
AX = 000Ah

**Return Registers:**
AX = maximum X coordinate in screen space (or character space if in text mode)

**See also:** Functions 000Bh and 0045h

**INT 62h Function 000Bh** *FGDRIVER v1.10*
***"FG_GETMAXY" - GET MAXIMUM ROW IN SCREEN SPACE*** *(partially doc.)*

**Registers at call:**
AX = 000Bh

**Return Registers:**
AX = maximum Y coordinate in screen space (or character space if in text mode)

**See also:** Functions 000Ah and 0045h

**INT 62h Function 000Ch** *FGDRIVER v1.10*
***"FG_XALPHA" - CONVERT SCREEN COLUMN TO CHARACTER COLUMN*** *(partially doc.)*

**Registers at call:**
AX = 000Ch
BX = screen space column

**Return Registers:**
AX = character space column containing specified coordinate

**See also:** Functions 000Dh and 000Eh

**INT 62h Function 000Dh** *FGDRIVER v1.10*
***"FG_YALPHA" - CONVERT SCREEN ROW TO CHARACTER ROW*** *(partially doc.)*

**Registers at call:**
AX = 000Dh
BX = screen space row

**Return Registers:**
AX = character space row containing specified coordinate

**See also:** Functions 000Ch and 000Fh

**INT 62h Function 000Eh** *FGDRIVER v1.10*
***"FG_XCONVERT" - CONVERT CHARACTER COLUMN TO SCREEN COLUMN*** *(partially doc.)*

**Registers at call:**
AX = 000Eh
BX = character space column

**Return Registers:**
AX = screen space column of leftmost pixel in specified character col

**See also:** Functions 000Ch and 000Fh

**INT 62h Function 000Fh** *FGDRIVER v1.10*
***"FG_YCONVERT" - CONVERT CHARACTER ROW TO SCREEN ROW*** *(partially doc.)*

**Registers at call:**
AX = 000Fh
BX = character space row

**Return Registers:**
AX = screen space row of topmost pixel in specified character row

**See also:** Functions 000Dh and 000Eh

**INT 62h Function 0010h** *FGDRIVER v1.10*
***"FG_GETLINES" - GET TEXT ROWS FOR CURRENT VIDEO MODE*** *(partially doc.)*

**Registers at call:**
AX = 0010h

**Return Registers:**
AX = number of text rows on screen in current video mode

**See also:** Function 0011h

**INT 62h Function 0011h** *FGDRIVER v1.10*
***"FG_SETLINES" - SET TEXT ROWS ON SCREEN*** *(partially doc.)*

**Registers at call:**
AX = 0011h
BX = new screen size (25, 43, 50)

**Return Registers:**
none

**See also:** Function 0010h

### INT 62h Functions 0012h and 0013h

*FGDRIVER v1.10*

***UNUSED***

**Registers at call:**
AX = 0012h and 0013h

**Return Registers:**
AX = 0000h

### INT 62h Function 0014h

*FGDRIVER v1.10 (partially doc.)*

***"FG_DEFCOLOR" - ASSIGN COLOR VALUE TO COLOR INDEX***

**Registers at call:**
AX = 0014h
BX = color index (0000h-00FFh)
CX = new color value (0 to maximum color value for current video mode)

**Return Registers:**
none

**Details:** This call is ignored in text modes and 256-color graphics modes.

**See also:** Functions 0015h, 0016h, and 001Dh

### INT 62h Function 0015h

*FGDRIVER v1.10 (partially doc.)*

***"FG_GETCOLOR" - GET CURRENT TEXT ATTRIBUTE***

**Registers at call:**
AX = 0015h

**Return Registers:**
AX = current text attribute or color index (graphics modes)

**See also:** Functions 0019h and 001Ah

### INT 62h Function 0016h

*FGDRIVER v1.10 (partially doc.)*

***"FG_GETINDEX" - GET COLOR VALUE FOR COLOR INDEX***

**Registers at call:**
AX = 0016h
BX = color index (0000h to 00FFh)

**Return Registers:**
AX = color value for specified color index

**Details:** This call returns the value passed to it in text and 256-color graphics modes.

**See also:** Functions 0014h and 001Ch

### INT 62h Function 0017h

*FGDRIVER v1.10 (partially doc.)*

***"FG_PALETTE" - SET PALETTE / SET VIDEO DAC REGISTER***

**Registers at call:**
AX = 0017h
**---CGA 4-color graphics---**
BX = CGA paletee number
CX = background color
**---CGA 2-color graphics---**
BX ignored
CX = foreground color
**---16-color graphics---**
BX = palette register number
CX = palette value
**---256-color graphics---**
BX = DAC register number
CX = DAC value

**Return Registers:**
none

**Details:** This function is ignored in text modes and Hercules graphics modes.

Few EGA/VGA adapters correctly set the foreground color in CGA mode 6.

**See also:** Functions 0018h and 001Dh

### INT 62h Function 0018h

*FGDRIVER v1.10 (partially doc.)*

***"FG_PALETTES" - SET ALL PALETTE REGISTERS***

**Registers at call:**
AX = 0018h
ES:BX -> array of 16 WORDs

containing values for palette registers
(or first 16 DAC registers in
256-color modes)

**Return Registers:**
none

**Details:** This function is ignored in text modes, CGA and Hercules graphics modes.
**See also:** Function 0017h

---

### INT 62h Function 0019h
### *"FG_SETATTR" - SET TEXT-MODE CHARACTER ATTRIBUTE*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0019h
BX = foreground
CX = background
DX = blink (0000h nonblinking, 0001h blink)

**Return Registers:**
none

**Details:** This call is ignored in graphics modes.
**See also:** Functions 0015h, 001Ah, and 0037h

---

### INT 62h Function 001Ah
### *"FG_SETCOLOR" - SET CURRENT COLOR*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 001Ah
BX = new color index (or text attribute in text modes)

**Return Registers:**
none

**See also:** Functions 0015h and 0019h

---

### INT 62h Function 001Bh
### *"FG_SETRGB" - SET VIDEO DAC REGISTER CONTENTS*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 001Bh
BX = palette or DAC register number
CX = red color component
DX = green component
SI = blue component

**Return Registers:**
none

**Details:** The register number may be negative for Tandy, PCjr, and 200-line EGA graphics modes to specify an intense color.

This call has no effect in text, CGA graphics, and Hercules graphics modes.
**See also:** Functions 001Ch and 00A5h

---

### INT 62h Function 001Ch
### *"FG_GETRGB" - GET VIDEO DAC REGISTER CONTENTS*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 001Ch
ES:BX -> variable pointer record (see Table 52-2)
CX = DAC register number

**Return Registers:**
variables updated

**Details:** This call is ignored in text modes and CGA/EGA video modes (since they do not use DAC registers).
**See also:** Functions 001Bh, 001Dh, and 00A4h

**52-2 Format of variable pointer record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of WORD buffer for red component of DAC register |
| 02h | WORD | offset of WORD buffer for red component |
| 04h | WORD | segment of WORD buffer for green component of DAC register |
| 06h | WORD | offset of WORD buffer for green component |
| 08h | WORD | segment of WORD buffer for blue component of DAC register |
| 0Ah | WORD | offset of WORD buffer for blue component |

---

**INT 62h Function 001Dh** *FGDRIVER v1.10 (partially doc.)*

### *"FG_MAPRGB" - MAP COLOR COMPONENTS INTO PALETTE VALUE*

**Registers at call:**
AX = 001Dh
BX = red component
CX = green component
DX = blue component

**Return Registers:**
AX = mode-specific palette value corresponding to specified components

**Details:** This call is meaningful only in 16-color graphics modes.

**See also:** Functions 0015h and 0017h

---

**INT 62h Function 001Eh** *FGDRIVER v1.10 (partially doc.)*

### *"FG_ERASE" - CLEAR THE ACTIVE VIDEO PAGE*

**Registers at call:**
AX = 001Eh

**Return Registers:**
none

**Details:** This function sets each pixel to 0 in graphics modes, each character cell to a blank with a gray foreground attribute in text modes.

---

**INT 62h Function 001Fh** *FGDRIVER v1.10 (partially doc.)*

### *"FG_POINT" - DISPLAY A PIXEL*

**Registers at call:**
AX = 001Fh
BX = column
CX = row

**Return Registers:**
none

**Details:** This call is ignored in text modes.

**See also:** Function 0020h, BIOS INT 10h Function 0Ch (Chapter 8)

---

**INT 62h Function 0020h** *FGDRIVER v1.10 (partially doc.)*

### *"FG_GETPIXEL" - GET COLOR OF SPECIFIED PIXEL*

**Registers at call:**
AX = 0020h
BX = column in screen space
CX = row in screen space

**Return Registers:**
AX = color value of pixel (0 to num_colors-1)
0000h in text modes

**See also:** Function 001Fh, BIOS INT 10h Function 0Dh (Chapter 8)

---

**INT 62h Functions 0021h and 0022h** *FGDRIVER v1.10 (partially doc.)*

### *"FG_GETXPOS" - GET GRAPHICS CURSOR ROW/COLUMN*

**Registers at call:**
AX = 0021h get column
AX = 0022h get row

**Return Registers:**
AX = screen space X/Y coordinate of graphics cursor position

**See also:** Functions 0023h and 0024h

---

**INT 62h Function 0023h** *FGDRIVER v1.10 (partially doc.)*

### *"FG_MOVE" - SET GRAPHICS CURSOR POSITION*

**Registers at call:**
AX = 0023h
BX = new column
CX = new row

**Return Registers:**
none

**Details:** This call is ignored in text modes.

**See also:** Functions 0021h, 0022h, and 0024h

---

## INT 62h Function 0024h

## "FG_MOVEREL" - ADJUST GRAPHICS CURSOR POSITION

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0024h
BX = column offset
CX = row offset

**Return Registers:**
none

**Details:** This call is ignored in text modes.

**See also:** Functions 0021h, 0022h, and 0023h

---

## INT 62h Functions 0025h and 0026h

## "FG_DRAW","FG_DRAWREL" - DRAW SOLID LINE

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = function
  0025h draw to absolute position
  0026h draw to relative position
BX = endpoint column or column offset
CX = endpoint row or row offset

**Return Registers:**
none

**Details:** These calls are ignored in text modes.

The starting point is the current graphics cursor position; the cursor position is updated.

**See also:** Functions 0027h and 0028h

---

## INT 62h Functions 0027h and 0028h

## "FG_DASH","FG_DASHREL" - DRAW DASHED LINE

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = function
  0027h draw to absolute position
  0028h draw to relative position
BX = endpoint column or column offset
CX = endpoint row or row offste
DX = dash pattern (set bits cause drawn pixels)

**Return Registers:**
none

**Details:** These calls are ignored in text modes.

The starting point is the current graphics cursor position; the cursor position is updated.

**See also:** Functions 0025h and 0026h

---

## INT 62h Function 0029h

## "FG_SETCLIP" - SET CLIPPING REGION

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0029h
BX = left edge of clipping region
CX = right edge of clipping region
DX = top edge of clipping region
SI = bottom edge of clipping region

**Return Registers:**
none

**See also:** Functions 004Ah, 004Ch, 0052h, and 0054h

---

## INT 62h Function 002Ah

## "FG_RECT" - DRAW UNFILLED RECTANGLE IN SCREEN SPACE

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 002Ah
BX = left edge column
CX = right edge column
DX = top edge row
SI = bottom edge row

**Return Registers:**
none

**See also:** Functions 002Bh, 002Ch, 002Eh, and 00A2h

---

**INT 62h Function 002Bh** FGDRIVER v1.10 *(partially doc.)*

***"FG_CLPRECT" - DRAW FILLED RECTANGLE IN SCREEN SPACE***

**Registers at call:**
AX = 002Bh
BX = screen space column of left edge
CX = screen space column of right edge
DX = screen space row of top edge
SI = screen space row of bottom edge

**Return Registers:**
none

**Details:** This call is ignored in text modes.

**See also:** Functions 002Ah, 002Ch, and 00A2h

---

**INT 62h Function 002Ch** FGDRIVER v1.10 *(partially doc.)*

***"FG_DRECT" - DRAW DITHERED RECTANGLE IN SCREEN SPACE***

**Registers at call:**
AX = 002Ch
BX = screen space column of left edge
CX = screen space column of right edge
DX = screen space row of top edge
SI = screen space row of bottom edge
ES:DI -> dithering matrix (video-mode dependent)

**Return Registers:**
none

**Details:** This call is ignored in text modes.

**See also:** Functions 002Bh and 00A2h

---

**INT 62h Function 002Dh** FGDRIVER v1.10 *(partially doc.)*

***"FG_ELLIPSE" - DRAW UNFILLED ELLIPSE IN SCRN SPACE***

**Registers at call:**
AX = 002Dh
BX = horizontal semi-axis length in screen space units
CX = vertical semi-axis length in screen space units

**Return Registers:**
none

**Details:** This call is ignored in text modes. The ellipse is centered at the current graphics cursor position.

**See also:** Functions 002Ah, 002Eh, and 00A2h

---

**INT 62h Function 002Eh** FGDRIVER v1.10 *(partially doc.)*

***"FG_POLYGON" - DRAW AN UNFILLED POLYGON***

**Registers at call:**
AX = 002Eh
CX = number of vertices in polygon
ES:BX -> variable pointer record (see Table 52-3)

**Return Registers:**
none

**See also:** Functions 002Dh, 0030h, and 00A2h

**52-3 Format of variable pointer record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of WORD array containing vertex columns |
| 02h | WORD | offset of WORD array containing vertex columns |
| 04h | WORD | segment of WORD array containing vertex rows |
| 06h | WORD | offset of WORD array containing vertex rows |

---

**INT 62h Function 002Fh** FGDRIVER v1.10 *(partially doc.)*

***"FG_PAINT" - FLOOD CLOSED REGION WITH COLOR***

**Registers at call:**
AX = 002Fh

BX = column
CX = row

**Return Registers:**
none

**Details:** This function fills an arbitrary closed region around the specified point with the current color; the screen edges are not considered region boundaries.

This call is ignored in text modes.

---

### INT 62h Function 0030h

***"FG_CIRCLE" - DRAW UNFILLED CIRCLE***

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0030h
BX = radius in horizontal screen space units (> 0)

**Return Registers:**
none

**Details:** The circle is drawn in screen space, centered at the current graphics cursor position.

This call is ignored in text modes.

**See also:** Function 00A2h

---

### INT 62h Function 0031h

***UNUSED***

*FGDRIVER v1.10*

**Registers at call:**
AX = 0031h

**Return Registers:**
AX = 0000h

---

### INT 62h Function 0032h

***"FG_TEXT" - DISPLAY STRING OF CHARACTERS***

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0032h
CX = length of string
ES:BX -> string

**Return Registers:**
none

**Details:** The string is displayed staring at the text cursor position using the current text attribute (text modes) or color index (graphics modes).

The text cursor position is updated after this call.

**See also:** Functions 001Ah, 0033h, 0035h, and 0036h

---

### INT 62h Function 0033h

***"FG_LOCATE" - SET TEXT-MODE CURSOR POSITION***

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0033h
BX = row
CX = column

**Return Registers:**
none

**Details:** There are only eight text cursors shared by successive groups of eight video pages (pages 0, 8, 16, ... share one cursor, 1, 9, ... share the second, etc).

**See also:** Function 0034h

---

### INT 62h Function 0034h

***"FG_WHERE" - GET CURRENT CURSOR POSITION***

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0034h
ES:BX -> variable pointers (see Table 52-4)

**Return Registers:**
indicated variables filled with cursor row and column for active display

**See also:** Function 0033h

**52-4 Format of variable pointers:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of WORD buffer for cursor row |
| 02h | WORD | offset of WORD buffer for cursor row |
| 04h | WORD | segment WORD buffer for cursor column |
| 06h | WORD | offset WORD buffer for cursor column |

---

### INT 62h Function 0035h

### *"FG_CHGATTR" - APPLY CURRENT TEXT ATTRIBUTE TO CHARACTERS*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0035h
BX = number of characters to recolor

**Return Registers:**
none

**Details:** This call is ignored in graphics modes.

Starting at the current text cursor position, the specified number of characters have their attributes changed to the current text attribute.

**See also:** Function 0036h

---

### INT 62h Function 0036h

### *"FG_CHGTEXT" - DISPLAY STRING AT CURSOR POSITION*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0036h
CX = length of string
ES:BX -> string to be displayed

**Return Registers:**
text cursor updated

**Details:** This call is ignored in graphics modes.

**See also:** Functions 0032h and 0035h

---

### INT 62h Function 0037h

### *"FG_GETATTR" - GET CHARACTER ATTRIBUTE FOR POSITION*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0037h
BX = row
CX = column

**Return Registers:**
AX = character attribute at specified location on active video page

**Details:** This call is ignored in graphics modes.

**See also:** Functions 0019h and 0038h

---

### INT 62h Function 0038h

### *"FG_GETCHAR" - GET CHARACTER FOR SCREEN POSITION*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0038h
BX = row
CX = column

**Return Registers:**
AX = character at specified location on active video page

**Details:** This call is ignored in graphics modes.

**See also:** Function 0037h

---

### INT 62h Functions 0039h to 003Bh

### *UNUSED*

*FGDRIVER v1.10*

**Registers at call:**
AX = 0039h to 003Bh

**Return Registers:**
AX = 0000h

---

### INT 62h Function 003Ch

### *"FG_GETPAGE" - GET ACTIVE VIDEO PAGE NUMBER*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 003Ch

**Return Registers:**
AX = active video page (0000h-003Fh)

**See also:** Functions 003Dh, 003Eh, and 0040h

---

### INT 62h Function 003Dh

### *"FG_SETPAGE" - SET ACTIVE VIDEO PAGE*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 003Dh
BX = new video page (0000h to 003Fh)

**Return Registers:**
none

**Details:** The specified page must be a physical or virtual page.
**See also:** Functions 0001h, 003Ch, 003Fh, and 0041h

---

### INT 62h Function 003Eh
*FGDRIVER v1.10*
*(partially doc.)*

### *"FG_GETVPAGE" - GET VISIBLE VIDEO PAGE NUMBER*

| Registers at call: | Return Registers: |
|---|---|
| AX = 003Eh | AX = visible video page (0000h-003Fh) |

**See also:** Functions 003Ch, 003Fh, and 0040h

---

### INT 62h Function 003Fh
*FGDRIVER v1.10*
*(partially doc.)*

### *"FG_SETVPAGE" - SET VISIBLE VIDEO PAGE*

| Registers at call: | Return Registers: |
|---|---|
| AX = 003Fh | none |
| BX = new video page (0000h to 003Fh) | |

**Details:** The specified page must be a physical or virtual page.
**See also:** Functions 0001h, 003Dh, 003Eh, and 0041h

---

### INT 62h Function 0040h
*FGDRIVER v1.10*
*(partially doc.)*

### *"FG_GETHPAGE" - GET CURRENT HIDDEN VIDEO PAGE NUMBER*

| Registers at call: | Return Registers: |
|---|---|
| AX = 0040h | AX = current hidden video page number (0000h-003Fh) |

**See also:** Functions 003Ch, 003Eh, and 0041h

---

### INT 62h Function 0041h
*FGDRIVER v1.10*
*(partially doc.)*

### *"FG_SETHPAGE" - SET HIDDEN VIDEO PAGE*

| Registers at call: | Return Registers: |
|---|---|
| AX = 0041h | none |
| BX = new hidden page (0000h to 003Fh) | |

**Details:** The specified page must be a physical page or a virtual page.
**See also:** Functions 003Dh, 003Fh, and 0040h

---

### INT 62h Function 0042h
*FGDRIVER v1.10*
*(partially doc.)*

### *"FG_ALLOCATE" - CREATE VIRTUAL VIDEO PAGE*

| Registers at call: | Return Registers: |
|---|---|
| AX = 0042h | AX = status (0000h,0001h,0007h,0008h) (see Table 52-5) |
| BX = page number (0000h-003Fh) | |

**Details:** The amount of memory required by the virtual video page depends on the current video mode.
**See also:** Functions 0043h, 009Dh, and 009Fh

**52-5 Values for status:**

| | |
|---|---|
| 0000h | successful |
| 0001h | specified page is a physical or logical page |
| 0007h | virtual page created/released, but memory control blocks corrupted |
| 0008h | not enough memory |
| 0009h | attempt to free a page which was never created |

---

### INT 62h Function 0043h
*FGDRIVER v1.10*
*(partially doc.)*

### *"FG_FREEPAGE" - FREE VIRTUAL OR LOGICAL VIDEO PAGE*

| Registers at call: | Return Registers: |
|---|---|
| AX = 0043h | AX = status (0000h,0001h,0007h,0009h) (see Table 52-5) |
| BX = page number (0000h-003Fh) | |

**See also:** Functions 0042h, 009Dh, 009Eh, and 009Fh

### INT 62h Function 0044h
### *"FG_GETADDR" - GET SEGMENT OF ACTIVE VIDEO PAGE*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0044h

**Return Registers:**
AX = segment of active video page

### INT 62h Function 0045h
### *"FG_RESIZE" - SET GRAPHICS MODE VIDEO PAGE SIZE*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0045h
BX = new page width in pixels
CX = new page height in pixels

**Return Registers:**
none

**Details:** The visible page must be set to 0000h before making this call.

Mouse, joysticks, expanded memory, and extended memory must be reinitialized after this call.

**See also:** Functions 000Ah, 000Bh, 0075h, 007Ah, 00A0h, and 00A1h

### INT 62h Functions 0046h and 0047h
### *"FG_GETMAP","FG_DRWMAP" - MODE-INDEPENDENT BITMAPS*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = function
    0046h store image as mode-independent bitmap
    0047h display mode-independent bitmap
ES:BX -> (buffer for) video mode-independent bitmap
CX = width of bitmap in bytes
DX = height of bitmap in pixel rows

**Return Registers:**
Function 0046h: each bit in bitmap is set if corresponding pixel is of the current color, cleared otherwise
Function 0047h: each set bit is drawn in the current color

**Details:** The lower left corner of the image's position on the screen is specified by the current cursor position (either text or graphics).

These functions are ignored in text modes.

**See also:** Functions 0048h, 0049h, 004Ah, 004Dh, and 0052h

### INT 62h Functions 0048h to 004Ch
### *STORE/DISPLAY BITMAPPED IMAGE*

*FGDRIVER v1.10*
*(part. doc.)*

**Registers at call:**
AX = function
    0048h "FG_GETIMAGE" store image as bitmap
    0049h "FG_DRWIMAGE" display bitmapped image
    004Ah "FG_CLPIMAGE" display clipped image
    004Bh "FG_REVIMAGE" display reversed image
    004Ch "FG_FLPIMAGE" display inverted clipped image
ES:BX -> (buffer for) video mode-specific bitmap
CX = width of bitmap in bytes
DX = height of bitmap in pixel rows

**Return Registers:**
none

**Details:** The lower left corner of the image's position on the screen is specified by the current cursor position (either text or graphics).

Functions 004Ah to 004Ch are ignored in text modes.

**See also:** Functions 0029h, 0046h, 0047h, 0052h, and 0062h

---

### INT 62h Function 004Dh
FGDRIVER v1.10

### *"FG_DISPLAY" - DISPLAY IMAGE (STD PIXEL RUN FORMAT)*
*(partially doc.)*

**Registers at call:**
AX = 004Dh
ES:BX -> pixel run map (pairs of bytes: color index, count)
CX = number of pixel runs to display
DX = width of image in pixels (> 0)

**Return Registers:**
none

**Details:** This call is ignored in text modes.

The image is displayed with its lower left corner at the current graphics cursor position.

**See also:** Functions 0047h, 004Eh, 004Fh, 0050h, and 0060h

---

### INT 62h Function 004Eh
FGDRIVER v1.10

### *"FG_DISPLAY" - DISPLAY IMAGE (PACKED PIXEL RUN FORMAT)*
*(partially doc.)*

**Registers at call:**
AX = 004Eh
ES:BX -> pixel run map
CX = number of pixel runs to display
DX = width of image in pixels (> 0)

**Return Registers:**
none

**Details:** This call is ignored in text modes.

The packed pixel run map consists of trios of bytes: colors, count1, count2; *colors* contains the color for the first run in its high nybble and the color for the second run in its low nybble.

The image is displayed with its lower left corner at the current graphics cursor position.

**See also:** Functions 004Dh, 004Fh, 0050h, and 0060h

---

### INT 62h Function 004Fh
FGDRIVER v1.10

### *"FG_DISPFILE" - DISPLAY STORED IMAGE*
*(partially doc.)*

**Registers at call:**
AX = 004Fh
ES:BX -> ASCIZ filename
CX = image width in pixels (> 0)
DX = image format
  0000h Fastgraph standard pixel run format
  0001h packed pixel run format

**Return Registers:**
none

**Details:** This call is ignored in text modes.

The image is displayed with its lower left corner at the current graphics cursor position.

**See also:** Functions 004Dh, 004Eh, 0050h, and 0060h

---

### INT 62h Function 0050h
FGDRIVER v1.10

### *"FG_PATTERN" - SPECIFY DISPLAY PATTERN FOR COLOR*
*(partially doc.)*

**Registers at call:**
AX = 0050h
BX = index of pattern to define
CX = number of predefined display pattern

**Return Registers:**
none

**Details:** When displaying a pixel run map, Fastgraph uses the pattern associated with each color index rather than displaying the actual color.

This call has no effect in text and 256-color graphics modes.

**See also:** Functions 004Dh, 004Eh, and 004Fh

---

### INT 62h Functions 0051h to 0054h
### *DISPLAY IMAGE STORED AS MASKING MAP*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = function
0051h "FG_DRAWMASK" normal image
0052h "FG_CLIPMASK" clipped image
0053h "FG_REVMASK" reversed image
0054h "FG_FLIPMASK" inverted image
ES:BX -> array containing image stored as a masking map (see Table 52-6)
CX = number of pixel runs in masking map
DX = width of masking map in pixels

**Return Registers:**
none

**Details:** These calls are ignored in text modes and in native EGA and VGA graphics modes.

The image is drawn with its lower left corner at the current graphics cursor position.

**See also:** Functions 0029h, 004Ah, and 0054h

**52-6 Format of masking map:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | length of first "protect" run (pixels remain unchanged) |
| 01h | BYTE | length of first "zero" run (pixels set to background color) |
| 02h | BYTE | length of second "protect" run |
| 03h | BYTE | length of second "zero" run |
| | ... | |

---

### INT 62h Functions 0055h to 0059h
### *UNUSED*

*FGDRIVER v1.10*

**Registers at call:**
AX = 0055h to 0059h

**Return Registers:**
AX = 0000h

---

### INT 62h Functions 005Ah and 005Bh
### *COPY REGION BETWEEN VISIBLE AND HIDDEN PAGE*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = direction of copy
005Ah "FG_SAVE" visible to hidden page
005Bh "FG_RESTORE" hidden to visible page
DX,BX = row,column of upper left corner
SI,CX = row,column of lower right corner

**Return Registers:**
none

**Details:** The left and right edges are adjusted to byte boundaries if necessary.

**See also:** Functions 005Ch and 0064h

---

### INT 62h Function 005Ch
### *"FG_TRANSFER" - COPY REGION*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 005Ch
CX = source video page
DX = destination video page
ES:BX -> copy record (see Table 52-7)

**Return Registers:**
none

**Details:** The source and destination regions must not overlap if they are on the same page.

**See also:** Functions 005Ah, 005Bh, 005Dh and 005Fh

**52-7 Format of copy record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | left edge column of source region |
| 02h | WORD | right edge column of source region |

| | | |
|---|---|---|
| 04h | WORD | top edge row of source region |
| 06h | WORD | bottom edge row of source region |
| 08h | WORD | left edge of destination |
| 0Ah | WORD | bottom edge of destination |

---

### INT 62h Function 005Dh
### *"FG_TCXFER" - COPY REGION EXCLUDING TRANSPARENT*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 005Dh
CX = source video page
DX = destination video page
ES:BX -> copy record (see Table 52-7)

**Return Registers:**
none

**Details:** Pixels which are in any of the colors defined as transparent with Function 005Eh are left unchanged in the destination region.

The source and destination regions must not overlap if they are on the same page.

This call is ignored in text modes.

**See also:** Functions 005Ch and 005Eh

---

### INT 62h Function 005Eh
### *"FG_TCMASK" - SET TRANSPARENT COLORS*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 005Eh
BX = colors to consider transparent (bit 0 = color 0, etc)

**Return Registers:**
none

**Details:** This call is ignored in text modes.

The specified colors are considered transparent by Function 005Dh.

**See also:** Function 005Dh

---

### INT 62h Function 005Fh
### *"FG_COPYPAGE" - TRANSFER VIDEO PAGE CONTENTS*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 005Fh
BX = source page number (0000h-003Fh)
CX = destination page number (0000h-003Fh)

**Return Registers:**
none

**Details:** If both source and destination pages are logical pages, they must both be located in the same type (conventional, EMS, XMS) of memory.

**See also:** Functions 005Ch, 009Dh, 009Eh, and 009Fh

---

### INT 62h Function 0060h
### *"FG_DISPPCX" - DISPLAY PCX FILE*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0060h
ES:BX -> ASCIZ filename
CX = flags
    bit 0: use current palette rather than PCX file's palette
    bits 1-15 reserved (0)

**Return Registers:**
AX = status
    0000h success
    0001h file not found
    0002h file is not a valid PCX file

**Details:** This call is ignored in text modes and Hercules low-resolution graphics.

The image is displayed with its upper left corner at the current graphics cursor position.

**See also:** Functions 004Dh, 004Eh, 004Fh, and 0061h

---

### INT 62h Function 0061h
### *"FG_MAKEPCX" - CREATE PCX FILE FROM SCREEN WINDOW*

*FGDRIVER v1.10 (partially doc.)*

**Registers at call:**
AX = 0061h
BX = left edge in screen space units
CX = right edge in screen space units
DX = top edge in screen space units
SI = bottom edge in screen space units
ES:DI -> ASCIZ filename of PCX file to create

**Return Registers:**
AX = status
0000h successful
0001h file not created

**Details:** The PCX file is created from the specified region of the active video page; the left and right edges are adjusted to a byte boundary if necessary.

If the specified file already exists, it is overwritten.

This call is ignored in text and Hercules low-resolution graphics modes.

**See also:** Function 0060h

---

### INT 62h Function 0062h
### *"FG_IMAGESIZ" - DETERMINE IMAGE STORAGE REQUIREMENT*

*FGDRIVER v1.10 (partially doc.)*

**Registers at call:**
AX = 0062h
BX = image width in pixels
CX = image height in pixels

**Return Registers:**
DX:AX = size in bytes of mode-specific bitmap for current video mode

**See also:** Function 0048h

---

### INT 62h Function 0063h
### *UNUSED*

*FGDRIVER v1.10*

**Registers at call:**
AX = 0063h

**Return Registers:**
AX = 0000h

---

### INT 62h Functions 0064h and 0065h
### *"FG_FADEIN","FG_FADEOUT" - FADE DISPLAY*

*FGDRIVER v1.10 (partially doc.)*

**Registers at call:**
AX = function
0064h fade-in hidden page
0065h fade-out to current color
BX = delay (0000h = fastest possible fade)

**Return Registers:**
none

**Details:** When fading in, the current hidden page is copied to the current visible page in small random sections to produce a fade-in effect; similarly, the visible page is filled with pixels of the current color in small random sections for a fade-out.

This call is ignored in text modes.

**See also:** Function 005Bh

---

### INT 62h Function 0066h
### *"FG_PAN" - SET SCREEN ORIGIN*

*FGDRIVER v1.10 (partially doc.)*

**Registers at call:**
AX = 0066h
BX = new column for screen origin
CX = new row for screen origin

**Return Registers:**
none

**See also:** Function 0067h

---

### INT 62h Function 0067h
### *"FG_SCROLL" - VERTICALLY SCROLL SCREEN REGION*

*FGDRIVER v1.10 (partially doc.)*

**Registers at call:**
AX = 0067h

**Return Registers:**
none

BX = left edge column
CX = right edge column
DX = top edge row
SI = bottom edge row
DI = number of pixels by which to scroll (positive scrolls up, negative scrolls down)
ES = type of scroll
  0000h circular (rows scrolled off are copied to vacated rows)
  else vacated rows are filled with the current color

**Details:** In graphics modes, the left and right edges are adjusted to byte boundaries if necessary. Circular scrolling uses part of the hidden page as a workspace.

**See also:** Function 0066h

---

### INT 62h Functions 0068h to 006Ch
### *UNUSED*

*FGDRIVER v1.10*

**Registers at call:**
AX = 0068h to 006Ch

**Return Registers:**
AX = 0000h

---

### INT 62h Function 006Dh
### *"FG_WAITKEY" - FLUSH KEYBOARD BUFFER AND AWAIT KEY*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 006Dh

**Return Registers:**
after next key pressed

**See also:** Functions 006Eh, 006Fh, and 0096h

---

### INT 62h Function 006Eh
### *"FG_GETKEY" - GET NEXT KEYSTROKE*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 006Eh
ES:BX -> variable pointer record (see Table 52-8)

**Return Registers:**
(after next keystroke if no typeahead)
  variables updated

**See also:** Functions 006Dh, 006Fh, and 0070h

**52-8 Format of variable pointer record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of BYTE buffer for ASCII keycode |
| 02h | WORD | offset of BYTE buffer for ASCII keycode |
| 04h | WORD | segment of BYTE buffer for extended keycode |
| 06h | WORD | offset of BYTE buffer for extended keycode |

---

### INT 62h Function 006Fh
### *"FG_INTKEY" - GET KEYSTROKE, NO WAIT*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 006Fh
ES:BX -> variable pointer record (see Table 52-8)

**Return Registers:**
variables updated

**Details:** If the keyboard buffer is empty, both the ASCII and extended keycodes are set to 00h.

**See also:** Functions 006Dh, 006Eh, 0070h, and 0079h

---

### INT 62h Functions 0070h and 0071h
### *"FG_CAPSLOCK","FG_SETCAPS" - GET/SET CAPSLOCK STATE*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = function
  0070h get current state

  0071h set state
    BX = new state (0000h off, 0001h on)

**Return Registers:**

AX = CapsLock state (0000h off, 0001h on)

none

**See also:** Functions 006Eh, 006Fh, 0072h, 0073h, and 0074h

### INT 62h Functions 0072h and 0073h
### *"FG_NUMLOCK","FG_SETNUM" - GET/SET NUMLOCK STATE*

*FGDRIVER v1.10*
*(partially doc.)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = function | |
| 0072h get currrent state | AX = NumLock state (0000h off, 0001h on) |
| 0073h set state<br>BX = new state (0000h off, 0001h on) | none |

**See also:** Functions 0070h, 0071h, and 0074h

### INT 62h Function 0074h
### *"FG_SCRLOCK" - GET STATE OF SCROLL LOCK KEY*

*FGDRIVER v1.10*
*(partially doc.)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 0074h | AX = ScrollLock state (0000h off, 0001h on) |

**See also:** Functions 0070h and 0072h

### INT 62h Function 0075h
### *"FG_INITJOY" - INITIALIZE JOYSTICK USE*

*FGDRIVER v1.10*
*(partially doc.)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 0075h | AX = status |
| BX = joystick number (0001h or 0002h) | 0000h successful<br>FFFFh joystick not connected or no game port |

**Details:** Fastgraph assumes that the requested joystick is centered at the time this function is called.

**See also:** Functions 0076h, 0077h, 0078h, and 0079h

### INT 62h Functions 0076h and 0077h
### *"FG_GETXJOY","FG_GETYJOY" - GET JOYSTICK POSITION*

*FGDRIVER v1.10*
*(partially doc.)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = function | AX = horizontal/vertical position of joystick |
| 0076h get horizontal position | FFFFh if joystick uninitialized or not present |
| 0077h get vertical position | |
| BX = joystick number (0001h or 0002h) | |

**Details:** The actual coordinates are processor- and joystick-dependent. You must call Function 0075h before this function.

**See also:** Functions 0075h and 0078h

### INT 62h Function 0078h
### *"FG_BUTTON" - GET JOYSTICK BUTTON STATE*

*FGDRIVER v1.10*
*(partially doc.)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 0078h | AX = button states |
| BX = joystick number (0001h or 0002h) | bit 0: top button pressed<br>bit 1: bottom button pressed |

**See also:** Functions 0075h, 0076h, 0077h, and 0079h

### INT 62h Function 0079h
### *"FG_INTJOY" - GET KEYCODES CORRESP TO JOYSTICK POSITION*

*FGDRIVER v1.10*
*(partially doc.)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 0079h | none |
| CX = joystick number (0001h or 0002h) | |
| ES:BX -> variable pointer record (see Table 52-9) | |

**Details:** If the indicated joystick has not been initialized with Function 0075h, both the button code and joystick position will be set to 00h.

If either button is pressed, a button code of 0Dh is returned; otherwise, a button code of 00h is returned.

**See also:** Functions 006Fh, 0075h, and 0078h

**52-9 Format of variable pointer record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of BYTE buffer for button code |
| 02h | WORD | offset of BYTE buffer for button code |
| 04h | WORD | segment of BYTE buffer for joystick position |
| 06h | WORD | offset of BYTE buffer for joystick position |

---

### INT 62h Function 007Ah
### *"FG_MOUSEINI" - INITIALIZE MOUSE SUPPORT*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 007Ah

**Return Registers:**
AX = status
0002h two-button mouse
0003h three-button mouse
FFFFh initialization failed

**Details:** After this call, the mouse cursor is invisible.

**See also:** Functions 007Bh, 007Ch, 007Eh, and 0080h; INT 33h Function 0000h (Chapter 21)

---

### INT 62h Function 007Bh
### *"FG_MOUSEVIS" - SET MOUSE CURSOR VISIBILITY*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 007Bh
BX = new state (0000h invisible, 0001h visible)

**Return Registers:**
none

**See also:** Functions 007Ah and 007Eh, INT 33h Functions 0001h and 0002h (Chapter 21)

---

### INT 62h Function 007Ch
### *"FG_MOUSEBUT" - GET MOUSE BUTTON PRESS/RELEASE COUNTS*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 007Ch
CX = mouse button (1 = left press,
2 = right press, 3 = middle press, negative for release)
ES:BX -> variable pointer record (see Table 52-10)

**Return Registers:**
variables updated

**Details:** Returns the count of presses or releases since the last call to this function; if the count is zero, row and col will also be zero.

**See also:** Function 007Ah, INT 33h Function 0005h, INT 33h Function 0006h

**52-10 Format of variable pointer record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of WORD buffer for press/release count |
| 02h | WORD | offset of WORD buffer for press/release count |
| 04h | WORD | segment of WORD buffer for screen space col of last press/rls |
| 06h | WORD | offset of WORD buffer for screen space col of last press/rels |
| 08h | WORD | segment of WORD buffer for screen space row of last press/rls |
| 0Ah | WORD | offset of WORD buffer for screen space row of last press/rls |

---

### INT 62h Function 007Dh
### *"FG_MOUSEPOS" - GET CURRENT MOUSE POSITION*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 007Dh
ES:BX -> variable pointer record (see Table 52-11)

**Return Registers:**
none

**See also:** Functions 007Ah and 0081h

**52-11 Format of variable pointer record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | segment of WORD buffer for mouse column |
| 02h | WORD | offset of WORD buffer for mouse column |
| 04h | WORD | segment of WORD buffer for mouse row |
| 06h | WORD | offset of WORD buffer for mouse row |
| 08h | WORD | segment of WORD buffer for button status |
| 0Ah | WORD | offset of WORD buffer for button status |

**Note:** Button status: bit 0 = left button, bit 1 = right, bit 2 = middle

---

### INT 62h Function 007Eh
### *"FG_MOUSECUR" - SPECIFY TEXT-MODE MOUSE CURSOR*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 007Eh
BX = screen mask
CX = cursor mask

**Return Registers:**
none

**Details:** This call is ignored in graphics modes.

**See also:** Functions 007Ah and 007Fh, INT 33h Function 000Ah (Chapter 21)

---

### INT 62h Function 007Fh
### *"FG_MOUSEPTR" - SPECIFY GRAPH-MODE MOUSE CURSOR*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 007Fh
ES:BX -> masks (16-byte screen mask followed by 16-byte cursor mask)
CX = X offset of hot spot from upper left corner
DX = Y offset of hot spot from upper left corner

**Return Registers:**
none

**Details:** This call is ignored in text modes.

**See also:** Functions 007Ah and 007Eh, INT 33h Function 0009h (Chapter 21)

---

### INT 62h Function 0080h
### *"FG_MOUSELIM" - SPECIFY MOUSE CURSOR LIMITS*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0080h
BX = left-most position allowed for mouse cursor
CX = right-most position allowed
DX = top-most position allowed
SI = bottom-most position allowed

**Return Registers:**
none

**See also:** Functions 007Ah and 0081h, INT 33h Functions 0007h and 0008h (Chapter 21)

---

### INT 62h Function 0081h
### *"FG_MOUSEMOV" - SET MOUSE CURSOR POSITION*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0081h
BX = new column
CX = new row

**Return Registers:**
none

**Details:** This function will not move the mouse cursor outside the bounding box specified with Function 0080h.

**See also:** Functions 007Ah, 007Dh, and 0080h; INT 33h Function 0004h (Chapter 21)

### INT 62h Function 0082h
### *"FG_MOUSESPD" - SET MOUSE CURSOR SPEED*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0082h
BX = horizontal mickeys per eight pixels of movement (default 16)
CX = vertical mickeys per eight pixels of movement (default 16)

**Return Registers:**
none

**See also:** INT 33h Functions 000Fh and 001Ah (Chapter 21)

### INT 62h Functions 0083h to 0087h
### *UNUSED*

*FGDRIVER v1.10*

**Registers at call:**
AX = 0083h to 0087h

**Return Registers:**
AX = 0000h

### INT 62h Function 0088h
### *"FG_SOUND" - MAKE SOUND FOR SPECIFIED DURATION*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0088h
BX = frequency in Hertz (18-32767)
CX = duration in clock ticks (0000h or negative for continuous sound)

**Return Registers:**
none

**Details:** This function is ignored if asynchronous sound (Functions 0089h, 008Bh, or 008Dh) is in progress.

**See also:** Functions 0089h, 008Ah, 008Eh, and 0090h; TopView INT 15h Function 1019h (Chapter 44)

### INT 62h Function 0089h
### *"FG_SOUNDS" - PLAY SOUNDS IN BACKGROUND*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 0089h
CX = number of times to cycle through sound list
ES:BX -> sounds array (see Table 52-12)

**Return Registers:**
none

**Details:** This function is ignored if asynchronous sound (Functions 0089h, 008Bh, or 008Dh) is in progress.

**See also:** Functions 0088h, 008Eh, and 008Fh; TopView INT 15h Function 1019h (Chapter 44)

**52-12 Format of sounds array element:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | frequency of sound in Hertz (0000h ends array) |
| 02h | WORD | duration of sound in clock ticks |

### INT 62h Function 008Ah
### *"FG_VOICE" - START SOUND*

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 008Ah
BX = channel on TI sound chip
1-3 = channels 1-3, 4 = channel 4 with periodic noise, 5 = channel 4 with white noise
CX = frequency in Hz (18-32767 for channels 1-3; 0=512 Hz, 1=1024 Hz, 2=2048 Hz for channels 4 and 5)
DX = volume
SI = duration in clock ticks (continuous if <= 0)

**Return Registers:**
none

**Details:** This function is only available on the PCjr and Tandy 1000 machines.

**See also:** Functions 0088h, 008Bh, and 008Ch

### INT 62h Function 008Bh
### "FG_VOICES" - PLAY SOUNDS IN BACKGROUND

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 008Bh
ES:BX -> tone array (see Table 52-13)
CX = number of times to repeat tone array

**Return Registers:**
none

**Details:** This function is only available on the PCjr and Tandy 1000 machines.
**See also:** Functions 008Ah, 008Dh, 008Eh, 008Fh, and 0091h

**52-13 Format of tone array element:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | channel number (0000h terminates array) |
| 02h | WORD | frequency |
| 04h | WORD | volume |
| 06h | WORD | duration in 1/72.8 seconds |

### INT 62h Function 008Ch
### "FG_MUSIC" - PLAY SERIES OF NOTES

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 008Ch
ES:BX -> '$'-terminated music string in BASIC PLAY format

**Return Registers:**
after music completed

**Details:** This function is ignored if asynchronous sound (Functions 0089h, 008Bh, or 008Dh) is in progress.
**See also:** Functions 008Ah and 008Dh

### INT 62h Function 008Dh
### "FG_MUSICB" - PLAY SERIES OF NOTES IN BACKGROUND

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 008Dh
CX = number of repetitions (negative = continuous play)
ES:BX -> '$'-terminated music string in BASIC PLAY format

**Return Registers:**
none

**Details:** This function is ignored if asynchronous sound (Functions 0089h, 008Bh, or 008Dh) is in progress.
**See also:** Functions 008Bh, 008Ch, 008Eh, 008Fh, and 0091h

### INT 62h Function 008Eh
### "FG_HUSH" - STOP ASYNCHRONOUS SOUND IMMEDIATELY

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 008Eh

**Return Registers:**
none

**Details:** This function immediately stops any sounds started with Functions 0089h, 008Bh, or 008Dh; it is ignored if no asynchronous sound is playing.
**See also:** Functions 0088h, 008Bh, 008Fh, 0090h, and 0091h

### INT 62h Function 008Fh
### "FG_HUSHNEXT" - STOP ASYNCHRONOUS SOUND

*FGDRIVER v1.10*
*(partially doc.)*

**Registers at call:**
AX = 008Fh

**Return Registers:**
none

**Details:** This function stops any sounds started with Functions 0089h, 008Bh, or 008Dh after the current repetition completes; it is ignored unless the asynchronous sound is continuous.
**See also:** Functions 008Bh, 008Eh, and 0090h

---

### INT 62h Function 0090h

*FGDRIVER v1.10*

***"FG_QUIET" - STOP CONTINUOUS SYNCHRONOUS SOUND***

*(partially doc.)*

**Registers at call:**
AX = 0090h

**Return Registers:**
none

**Details:** This call has no effect if there is no continuous sound playing.

**See also:** Functions 008Eh, 008Fh, and 0091h

---

### INT 62h Function 0091h

*FGDRIVER v1.10*

***"FG_PLAYING" - DETERMINE WHETHER ASYNC SOUND ACTIVE***

*(partially doc.)*

**Registers at call:**
AX = 0091h

**Return Registers:**
AX = sound state (0 = no asynchronous sound,
1 = async sound playing)

**See also:** Functions 008Bh, 008Dh, and 0090h

---

### INT 62h Functions 0092h and 0093h

*FGDRIVER v1.10*

***"FG_RESUME","FG_SUSPEND" - SUSPEND/RESTART ASYNCHRONOUS SOUND***

*(part. doc.)*

**Registers at call:**
AX = function
0092h restart sound
0093h temporarily stop sound

**Return Registers:**
none

**Details:** Function 0093h has no effect if there is no asynchronous sound in progress. The program must not exit while sound is suspended.

**See also:** Functions 008Eh, 008Fh, and 0090h

---

### INT 62h Functions 0094h and 0095h

*FGDRIVER v1.10*

***UNUSED***

**Registers at call:**
AX = 0094h and 0095h

**Return Registers:**
AX = 0000h

---

### INT 62h Functions 0096h and 0097h

*FGDRIVER v1.10*

***"FG_WAITFOR","FG_STALL" - DELAY FOR SPECIFIED DURATION***

*(partially doc.)*

**Registers at call:**
AX = time units
0096h clock ticks
0097h processor-dependent delay units
BX = duration

**Return Registers:**
after delay elapses

**See also:** Functions 006Dh and 0098h, INT 1Ah Function FF01h (Chapter 22), INT 2Fh Function 1224h (Chapter 24)

---

### INT 62h Function 0098h

*FGDRIVER v1.10*

***"FG_MEASURE" - GET DELAY UNITS PER CLOCK TICK***

*(partially doc.)*

**Registers at call:**
AX = 0098h

**Return Registers:**
AX = delay units per clock tick (processor-dependent)

**Details:** Delay units are used by "FG_STALL" (Function 0097h).

**See also:** Function 0097h

---

### INT 62h Function 0099h

*FGDRIVER v1.10*

***"FG_GETCLOCK" - GET CLOCK TICKS SINCE MIDNIGHT***

*(partially doc.)*

**Registers at call:**
AX = 0099h

**Return Registers:**
DX:AX = number of clock ticks since midnight

**See also:** BIOS INT 1Ah Function 00h (Chapter 4)

### INT 62h Function 009Ah
*FGDRIVER v1.10*

***UNUSED***

**Registers at call:**
AX = 009Ah

**Return Registers:**
AX = 0000h

### INT 62h Function 009Bh
*FGDRIVER v1.10*
*(partially doc.)*

***"FG_MEMAVAIL" - QUERY AMOUNT OF FREE MEMORY***

**Registers at call:**
AX = 009Bh

**Return Registers:**
DX:AX = number of bytes of conventional memory available

**See also:** Function 009Dh

### INT 62h Function 009Ch
*FGDRIVER v1.10*
*(partially doc.)*

***"FG_SETFUNC" - SET LOGICAL OPERATION FOR VIDEO OPS***

**Registers at call:**
AX = 009Ch
BX = operation (0000h replacement,
 0001h AND, 0002h OR, 0003h XOR)

**Return Registers:**
none

**Details:** This call is only available in native EGA/VGA graphics mode (0Dh to 12h).

### INT 62h Function 009Dh
*FGDRIVER v1.10*
*(partially doc.)*

***"FG_ALLOCCMS" - CREATE LOGICAL VIDEO PAGE (CONVMEM)***

**Registers at call:**
AX = 009Dh
BX = page number (0001h-003Fh)

**Return Registers:**
AX = status (0000h,FFFCh,FFFDh,FFFEh)
 (see Table 52-14)

**Details:** The only operation which is allowed on logical pages is Function 005Fh.

**See also:** Functions 0042h, 0043h, 005Fh, 009Bh, 009Eh, and 009Fh

**52-14 Values for status:**

| | |
|---|---|
| 0000h | successful |
| FFFCh | insufficient memory |
| FFFDh | page already created, or exists as physical or virtual page |
| FFFEh | invalid page number |
| FFFFh | memory manager not initialized |

### INT 62h Function 009Eh
*FGDRIVER v1.10*
*(partially doc.)*

***"FG_ALLOCEMS" - CREATE LOGICAL VIDEO PAGE (EMS)***

**Registers at call:**
AX = 009Eh
BX = page number (0001h-003Fh)

**Return Registers:**
AX = status (see Table 52-14)

**Details:** Must first call Function 00A0h.

The only operation which is allowed on logical pages is Function 005Fh.

**See also:** Functions 0043h, 005Fh, 009Dh, 009Fh, and 00A0h

### INT 62h Function 009Fh
*FGDRIVER v1.10*
*(partially doc.)*

***"FG_ALLOCXMS" - CREATE LOGICAL VIDEO PAGE (XMS)***

**Registers at call:**
AX = 009Fh
BX = page number (0001h-003Fh)

**Return Registers:**
AX = status (see Table 52-14)

**Details:** Must first call Function 00A1h.

The only operation which is allowed on logical pages is Function 005Fh.

**See also:** Functions 0043h, 005Fh, 009Dh, 009Eh, and 00A1h

---

**INT 62h Functions 00A0h and 00A1h** *FGDRIVER v1.10*

***"FG_INITEMS","FG_INITXMS" - INITIALIZE MEMORY USE*** *(partially doc.)*

**Registers at call:**
AX = function
    00A0h initialize expanded memory use
    00A1h initialize extended memory use

**Return Registers:**
AX = status
    0000h successful
    FFFFh memory manager inaccessible or not installed

**See also:** Functions 009Eh and 009Fh

---

**INT 62h Function 00A2h** *FGDRIVER v1.10*

***"FG_BOX" - DRAW UNFILLED RECTANGLE*** *(partially doc.)*

**Registers at call:**
AX = 00A2h
BX = left column
CX = right column
DX = top row
SI = bottom row

**Return Registers:**
none

**Details:** The rectangle is drawn in screen space, respecting the clipping region, with edges of the width specified with Function 00A3h (default = 1 is set by "FG_SETMODE").

This function has no effect in text modes.

**See also:** Functions 002Bh, 002Ch, 002Eh, 0030h, and 00A3h

---

**INT 62h Function 00A3h** *FGDRIVER v1.10*

***"FG_BOXDEPTH" - SET RECTANGLE BORDER WIDTH*** *(partially doc.)*

**Registers at call:**
AX = 00A3h
BX = width of left and right edges in pixels (> 0)
CX = width of top and bottom edges in pixels (> 0)

**Return Registers:**
none

**See also:** Function 00A2h

---

**INT 62h Functions 00A4h and 00A5h** *FGDRIVER v1.10*

***"FG_GETDACS","FG_SETDACS" - GET/SET VIDEO DAC CONTENTS*** *(partially doc.)*

**Registers at call:**
AX = function
    00A4h get DAC contents
    00A5h set DAC registers
CX = number of DAC registers (0001h to 0100h)
DX = starting DAC register number (0000h to 00FFh)
ES:BX -> buffer for/containing DAC red/green/blue triples

**Return Registers:**
none

**Details:** The register number wraps back to zero after FFh.

Function 00A5h has no effect in text modes or graphics modes below 11h.

**See also:** Functions 001Bh and 001Ch, BIOS INT 10h Functions 1012h and 1017h (Chapter 8)

## CHAPTER 53

# STSC APL*Plus/PC

APL is a mathematically-oriented programming language well-suited to manipulating vectors and matrices. In its original incarnation, it used the Greek alphabet and numerous special symbols as commands and operators. Such use of special symbols leads to compact but difficult-to-read code, and resulted in APL's reputation as a "write-only" language.

STSC, Inc. has been the major vendor of APL products on the IBM PC family for many years, and has produced a number of interpreters over the years. Some include special display fonts to provide the original symbol set, while others allow the use of keywords rather than symbols. In addition to interpreters, STSC provides such varies products based on APL as programming tools and a spreadsheet manager for interfacing with Lotus 1-2-3.

---

**INT 86h** *APL*PLUS/PC*

***Terminate APL session and return to DOS***

**Conflicts:** ROM BASIC (Chapter 54), NetBIOS (NI)

**See also:** INT 87h, INT C6h, MS-DOS INT 21h Function 4Ch (Chapter 24)

---

**INT 87h** *APL*PLUS/PC*

***UNDETERMINED FUNCTION***

**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)

**See also:** INT 86h, INT C7h, INT 88h Function 00h

---

**INT 88h Function 00h** *APL*PLUS/PC*

***CREATE OBJECT OF ARBITRARY RANK OR SHAPE***

**Registers at call:**

AL = 00h

BX = STPTR of the variable to be assigned

ES:SI -> model of type, rank, and shape (see Table 53-1)

**Return Registers:**

ES:DI -> first data byte of object

DX:CX = number of elements in the object

**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)

**See also:** INT C8h

**53-1 Format of shape model:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | type<br>01h character (2-byte dimension sizes)<br>02h integer (2-byte dimension sizes)<br>08h floating point (2-byte dimension sizes)<br>11h character (4-byte dimension sizes)<br>12h integer (4-byte dimension sizes) |

| | | |
|---|---|---|
| | | 18h floating point (4-byte dimension sizes) |
| 01h | BYTE | rank |
| 02h | WORD/DWORD | first dimension of shape |
| N | WORD/DWORD | second dimension of shape |
| | ... | |

---

### INT 88h Function 01h

_APL*PLUS/PC_

### _CREATE CHARACTER SCALAR/VECTOR/MATRIX <64K IN SIZE_

**Registers at call:**
AL = 01h
AH = rank
BX = STPTR of the variable to be assigned
CX = first dimension (if any)
DX = second dimension (if any)

**Return Registers:**
ES:DI -> object
CX = number of elements in the object

**Details:** Each dimension must be 32767 or smaller.
**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)
**See also:** Functions 02h and 08h, INT C8h

---

### INT 88h Function 02h

_APL*PLUS/PC_

### _CREATE INTEGER SCALAR/VECTOR/MATRIX <64K IN SIZE_

**Registers at call:**
AL = 02h
AH = rank
BX = STPTR of the variable to be assigned
CX = first dimension (if any)
DX = second dimension (if any)

**Return Registers:**
ES:DI -> object
CX = number of elements in the object

**Details:** Each dimension must be 32767 or smaller.
**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)
**See also:** Functions 01h and 08h, INT C8h

---

### INT 88h Function 08h

_APL*PLUS/PC_

### _CREATE FLOATING POINT SCALAR/VECTOR/MATRIX <64K IN SIZE_

**Registers at call:**
AL = 08h
AH = rank
BX = STPTR of the variable to be assigned
CX = first dimension (if any)
DX = second dimension (if any)

**Return Registers:**
ES:DI -> object
CX = number of elements in the object

**Details:** Each dimension must be 32767 or smaller.
**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)
**See also:** Functions 01h and 02h, INT C8h

---

### INT 88h Function F5h

_APL*PLUS/PC_

### _FORCE OBJECT INTO REAL WORKSPACE FROM VIRTUAL_

**Registers at call:**
AL = F5h
BX = STPTR of object

**Return Registers:**
none

**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)
**See also:** INT C8h

### INT 88h Function F6h and F7h

*APL*PLUS/PC*

#### *MAKE NAME IMMUNE FROM/ELIGIBLE FOR OUTSWAPPING*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AL = F6h make immune | none |
| AL = F7h make eligible | |
| BX = STPTR of object | |

**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)
**See also:** Function F8h, INT C8h

### INT 88h Function F8h

*APL*PLUS/PC*

#### *REPORT WHETHER NAME IS ELIGIBLE FOR OUTSWAPPING*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AL = F8h | BX = name's status |
| BX = STPTR of object | (0000h eligible, 0001h not eligible) |

**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)
**See also:** Functions F6h and F7h, INT C8h

### INT 88h Function F9h

*APL*PLUS/PC*

#### *DETERMINE NAME STATUS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AL = F9h | CF set if name ill-formed or already in use |
| ES:SI -> name | BX = STPTR if already in symbol table |
| CX = length of name | CF clear if name is available for use |
| | BX = 0000h |

**Details:** This function does not force the name into the workspace.
**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)
**See also:** Functions FEh and FFh, INT C8h

### INT 88h Function FCh

*APL*PLUS/PC*

#### *DETERMINE IF MEMORY AVAILABLE WITHOUT GARBAGE COLLECTION*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AL = FCh | CF clear if memory available |
| BX = amount of memory needed (paragraphs) | CF set if a workspace compaction is required |

**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)
**See also:** Function FDh, INT C8h

### INT 88h Function FDh

*APL*PLUS/PC*

#### *PERFORM GARBAGE COLLECTION AND RETURN AVAILABLE MEMORY*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AL = FDh | BX = number of paragraphs available in workspace |

**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)
**See also:** Function FCh, INT C8h

### INT 88h Function FEh

*APL*PLUS/PC*

#### *CREATE NAME*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AL = FEh | BX = STPTR of name |
| ES:SI -> name | DX = interpreter's data segment |
| CX = length of name | |

**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)
**See also:** Functions F9h and FFh, INT C8h

---

**INT 88h Function FFh** *APL*PLUS/PC*

***DETERMINE NAME STATUS***

| Registers at call: | Return Registers: |
|---|---|
| AL = FFh | CF set if name ill-formed or already in use |
| ES:SI -> name | BX = STPTR if already in symbol table |
| CX = length of name | CF clear if name is available for use |
| | BX = 0000h |

**Details:** This function forces the name into the workspace and makes it immune from outswapping.

**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)

**See also:** Functions F9h and FEh, INT C8h

---

**INT 8Ah** *APL*PLUS/PC*

***PRINT SCREEN***

**Details:** This call is the same as INT 05h.

**Conflicts:** ROM BASIC (Chapter 54)

**See also:** INT 8Ch, INT CAh, BIOS INT 05h (Chapter 4)

---

**INT 8Bh** *APL*PLUS/PC*

***BEEP***

**Details:** This call is the same as printing a ^G via INT 21h Function 02h.

**Conflicts:** ROM BASIC (Chapter 54), Virus (Chapter 59)

**See also:** INT CBh, MS-DOS INT 21h Function 02h (Chapter 24)

---

**INT 8Ch** *APL*PLUS/PC*

***CLEAR SCREEN MEMORY***

| Registers at call: | Return Registers: |
|---|---|
| AX = flag | none |
| 0000h do not save display attributes | |
| 0001h save attributes | |

**Conflicts:** ROM BASIC (Chapter 54)

**See also:** INT CCh

---

**INT 90h** *APL*PLUS/PC*

***USED BY PORT 10 PRINTER DRIVER***

**Conflicts:** ROM BASIC (Chapter 54)

---

**INT 95h** *APL*PLUS/PC*

***DETERMINE R= SPACE***

**Details:** Use this call only when the R= option is invoked on entering APL.

**Conflicts:** ROM BASIC (Chapter 54)

---

**INT A0h** *APL*PLUS/PC*

***USED BY APL/GSS*CGI GRAPHICS INTERFACE***

**Conflicts:** ROM BASIC (Chapter 54)

**See also:** GSS*CGI INT 59h (Chapter 8)

---

**INT C6h to INT CCh** *APL*PLUS/PC*

***IDENTICAL TO INT 86h THROUGH INT 8Ch***

**Details:** STSC moved its interrupts from 86h-8Ch to C6h-CCh, but did not delete the older interrupts.

**Conflicts:** ROM BASIC (Chapter 54)

**See also:** INT 86h through INT 8Ch

---

### INT CDh and INT CEh
*STSC APL*PLUS/PC*

***MAY BE USED IN FUTURE RELEASES***

**Details:** STSC has indicated that these interrupts may eventually be used.
**Conflicts:** ROM BASIC (Chapter 54)

---

### INT CFh
*APL*PLUS/PC*

***DEFAULT LOW-RESOLUTION TIMER FOR QUAD MF FUNCTION***

**Conflicts:** ROM BASIC (Chapter 54)
**See also:** INT E0h

---

### INT D0h to INT DCh
*STSC APL*PLUS/PC*

***MAY BE USED IN FUTURE RELEASES***

**Details:** STSC has indicated that these interrupts may eventually be used.
**Conflicts:** PC-MOS/386 (Chapter 49), ROM BASIC (Chapter 54), NJFRERAM (Chapter 66), Screen Thief (Chapter 66), PC/370 (NI)

---

### INT DDh
*STSC APL*PLUS/PC v9.0*

***PLACE KEYSTROKE EVENTS IN INPUT BUFFER***

| **Registers at call:** | **Return Registers:** |
|---|---|
| BX = where to place keystrokes<br>FFFFh insert before current buffer contents<br>0000h replace current contents<br>0001h insert after current contents<br>CX = number of keystroke events to insert<br>ES:SI -> data to be placed into buffer (list of WORD key codes)<br>4000h + N = normal ASCII keystroke N (N = 00h to FFh)<br>4100h + N = extended ASCII keystroke N (N = 03h to 84h) | none |

**Conflicts:** ROM BASIC (Chapter 54), Screen Thief (Chapter 66)
**See also:** BIOS INT 16h Function 05h (Chapter 4)

---

### INT DEh
*APL*PLUS/PC*

***UNDETERMINED FUNCTION***

**Details:** This interrupt appears to be the same as INT 16h.
**Conflicts:** ROM BASIC (Chapter 54), Screen Thief (Chapter 66)
**See also:** BIOS INT 16h (Chapter 4)

---

### INT DFh
*APL*PLUS/PC*

***SAME AS INT 10h***

**Conflicts:** Victor 9000 (Chapter 7), ROM BASIC (Chapter 54), Screen Thief (Chapter 66)
**See also:** BIOS INT 10h (Chapter 8)

---

### INT E0h
*APL*PLUS/PC*

***RESTIME HIGH-RESOLUTION TIMER FOR QUAD MF FUNCTION***

**Conflicts:** DR Multiuser DOS (Chapter 48), ROM BASIC (Chapter 54), Virus (Chapter 59)
**See also:** INT CFh

## CHAPTER 54

# *Programming Language Runtime Support*

Most high-level languages include a sizable runtime library of useful (and even essential) subroutines. A number of them use interrupts rather than direct calls to invoke or control functions in the runtime library. In this chapter, we cover Btrieve, floating-point emulation, overlay management, JPI TopSPEED Modula-2, Logitech Modula-2, SQL, Compaq's UILIB, Exact, and ROM BASIC, as well as the Borland C++ IDE.

---

**INT 11h Function 05D7h** — *Borland C++ IDE*
***INSTALLED CALLOUT*** — *Callout (undocumented)*

**Registers at call:**
AX = 05D7h
BX = product ID (0088h)

**Return Registers:**
none

**Details:** This fuction is called by the BC++ IDE when an application calls INT 12h Function 05D7h Subfunction 05D7h.
**Conflicts:** Alignment Check (Chapter 3), BIOS (Chapter 4), SDLP (Chapter 11)
**See also:** INT 12h Function 05D7h

---

**INT 12h Function 05D7h** — *Borland C++ IDE*
***INSTALLATION CHECK*** — *(undocumented)*

**Registers at call:**
AX = 05D7h (1495d)
BX = 05D7h

**Return Registers:**
none

**Details:** The BC++ IDE will call INT 11h Function 05D7h Subfunction 0088h if it is loaded.
**Conflicts:** BIOS (Chapter 4), KEYBUI (Chapter 13), PARKER (Chapter 66)
**See also:** INT 11h Function 05D7h Subfunction 0088h

---

**INT 16h Function AAh** — *PTxxx.COM*
***(xxx=CGA,EGA,VGA,HER...) CALL GATE FOR GRAPHICS***

**Registers at call:**
AH = AAh
various registers set up by high level language

**Return Registers:**
graphics operation performed

**Details:** PT stands for Paint Tools which is a graphics library for Turbo Pascal, Modula 2 and others from DataBiten in Sweden. The library is installed as a memory resident driver.

---

**INT 2Fh Function AB00h** — *Btrieve Multi-User*
***INSTALLATION CHECK***

**Registers at call:**
AX = AB00h

**Return Registers:**
AL = 4Dh if installed

**Conflicts:** none, but see also Table 2-2
**See also:** Functions AB01h and AB02h, Btrieve INT 7Bh

---

### INT 2Fh Function AB01h
*Btrieve Multi-User*

### *EXECUTE Btrieve OPERATION*

**Registers at call:**
AX = AB01h
BX = process ID
DS:DX -> 38-byte parameter record
(see Btrieve INT 7Bh)

**Return Registers:**
AL = status
00h OK
other retry after calling INT 7Fh Function 0200h

**See also:** Functions AB00h and AB02h, Btrieve INT 7Bh, Btrieve INT 7Fh Function 0200h

---

### INT 2Fh Function AB02h
*Btrieve Multi-User*

### *GET NEW PROCESS ID*

**Registers at call:**
AX = AB02h

**Return Registers:**
AL = 00h succesful
BX = process ID
AL > 00h failed, retry after calling INT 7Fh
Function 0200h

**See also:** Functions AB00h and AB01h, Btrieve INT 7Bh, Btrieve INT 7Fh Function 0200h

---

### INT 34h to INT 3Bh
*FLOATING POINT EMULATION*

### *OPCODES D8h to DFh*

**Purpose:** These interrupts are used to emulate floating-point instructions with opcodes of D8h through DFh (INT 34h is D8h, INT 35h is D9h, etc.). They use registers on entry and exit in the same manner as the instructions being emulated.
**Details:** The floating-point emulators in Borland and Microsoft languages and Lahey FORTRAN use these interrupts.
**Conflicts:** PC-MOS/386 INT 38h (Chapter 49)
**See also:** INT 3Eh

---

### INT 3Ch
*FLOATING POINT EMULATION*

### *INSTRUCTIONS WITH SEGMENT OVERRIDE*

**Details:** The floating-point emulators in Borland and Microsoft languages and Lahey FORTRAN use this interrupt.
The generated code is CD 3C xy mm .... where xy is a modified ESC instruction and mm is the modR/M byte. The xy byte appears to be encoded as

```
        s s 0 1 1 x x x
     or s s 0 0 0 x x x
```

where "ss" specifies the segment override: 00 = DS, 01 = SS, 10 = CS, and 11 = ES.
**See also:** INT 3Bh, INT 3Dh

---

### INT 3Dh
*FLOATING POINT EMULATION*

### *STANDALONE FWAIT*

**Details:** The floating-point emulators in Borland and Microsoft languages and Lahey FORTRAN use this interrupt.

This vector is modified but not restored by Direct Access v4.0, and may be left dangling by other programs written with the same version of compiled BASIC.
**See also:** INT 3Ch, INT 3Eh

### INT 3Eh *FLOATING POINT EMULATION*

### *Borland LANGUAGES "SHORTCUT" CALL*

**Details:** The two bytes following the INT 3Eh instruction are the subcode (see Table 54-1) and a NOP (90h), except for subcodes DCh and DEh, where the second byte is a register count (01h-08h).

This vector is modified but not restored by Direct Access v4.0, and may be left dangling by other programs written with the same version of compiled BASIC.

**See also:** INT 3Dh

**54-1 Values for subcode:**

| *Subcode* | *Function* |
|---|---|
| DCh | load 8086 stack with 8087 registers; overwrites the 10*N bytes at the top of the stack prior to the INT 3Eh with the 8087 register contents |
| DEh | load 8087 registers from top of 8086 stack; ST0 is furthest from top of 8086 stack |
| E0h | round TOS and R1 to single precision, compare, pop twice<br>returns AX=8087 status word, FLAGS=8087 condition bits |
| E2h | round TOS and R1 to double precision, compare, pop twice<br>returns AX=8087 status word, FLAGS=8087 condition bits |
| E4h | compare TOS/R1 with two POP's<br>returns FLAGS=8087 condition bits |
| E6h | compare TOS/R1 with POP<br>returns FLAGS=8087 condition bits |
| E8h | FTST (check TOS value)<br>returns FLAGS=8087 condition bits |
| EAh | FXAM (check TOS value)<br>returns AX=8087 status word |
| ECh | sine(ST0) |
| EEh | cosine(ST0) |
| F0h | tangent(ST0) |
| F2h | arctangent(ST0) |
| F4h | ST0 = ln(ST0) |
| F6h | ST0 = log2(ST0) |
| F8h | ST0 = log10(ST0) |
| FAh | ST0 = e**ST0 |
| FCh | ST0 = 2**ST0 |
| FEh | ST0 = 10**ST0 |

**Note:** Subcode E2h is buggy in TPas5.5, because it sets the 8087 precision control field to the undocumented value 01h; this results in actually rounding to single precision.

### INT 3Fh *Overlay manager interrupt (Microsoft LINK.EXE, Borland TLINK VROOMM)*

**Details:** INT 3Fh is the default, and may be overridden while linking.

This vector is modified but not restored by Direct Access v4.0, and may be left dangling by other programs written with the same version of compiled BASIC.

**See also:** OVERLAY INT FEh

### INT 3Fh *Microsoft Dynamic Link Library manager*

**See also:** INT 21h Function 4Bh

*SQL Base*

## INT 47h Function 8000h
### *DATABASE ENGINE API*

**Program:** SQL Base is a network-oriented database engine by Gupta Technologies.

**Registers at call:**
AX = 8000h
DS:BX -> parameter block, first word is function number (see Table 54-2)

**Return Registers:**
none

**Conflicts:** Acorn BBC Master (Chapter 7), Z100 (Chapter 7), Western Digital SuperBIOS (Chapter 10)

**See also:** Function 8001h

**54-2 Values for function number:**

| | |
|---|---|
| 01h | "SQLFINI" initalize application's use of the database |
| 02h | "SQLFDON" application is done using the database |
| 03h | "SQLFCON" connect to a cursor/database |
| 04h | "SQLFDIS" disconnect from a cursor/database |
| 05h | "SQLFCOM" compile a SQL command |
| 06h | "SQLFEXE" execute a SQL command |
| 07h | "SQLFCEX" compile and execute a SQL command |
| 08h | "SQLFCMT" commit a transaction to the database |
| 09h | "SQLFDES" describe the items of a SELECT statement |
| 0Ah | "SQLFGFI" get fetch information |
| 0Bh | "SQLFFBK" fetch previous result row from SELECT statement |
| 0Ch | "SQLFFET" fetch next result row from SELECT statement |
| 0Dh | "SQLFEFB" enable fetch backwards |
| 0Eh | "SQLFPRS" position in result set |
| 0Fh | "SQLFURS" undo result set |
| 10h | "SQLFNBV" get number of bind variables |
| 11h | "SQLFBND" bind data variables |
| 12h | "SQLFBNN" bind numerics |
| 13h | "SQLFBLN" bind long number |
| 14h | "SQLFBLD" bind long data variables |
| 15h | "SQLFSRS" start restriction set processing |
| 16h | "SQLFRRS" restart restriction set processing |
| 17h | "SQLFCRS" close restriction set |
| 18h | "SQLFDRS" drop restriction set |
| 19h | "SQLFARF" apply Roll Forward journal |
| 1Ah | "SQLFERF" end Roll Forward journal |
| 1Bh | "SQLFSRF" start Roll Forward journal |
| 1Ch | "SQLFSTO" store a compiled SQL command |
| 1Dh | "SQLFRET" retrieve a compiled SQL command |
| 1Eh | "SQLFDST" drop a stored command |
| 1Fh | "SQLFCTY" get command type |
| 20h | "SQLFEPO" get error position |
| 21h | "SQLFGNR" get number of rows |
| 22h | "SQLFNSI" get number of select items |
| 23h | "SQLFRBF" get Roll Back flag |
| 24h | "SQLFRCD" get return code |
| 25h | "SQLFROW" get number of ROWs |
| 26h | "SQLFSCN" set cursor name |

| | |
|---|---|
| 27h | “SQLFSIL” set isolation level |
| 28h | “SQLFSLP” set log parameters |
| 29h | “SQLFSSB” set select buffer |
| 2Ah | “SQLFSSS” set sort space |
| 2Bh | “SQLFRLO” read long |
| 2Ch | “SQLFWLO” write long |
| 2Dh | “SQLFLSK” long seek |
| 2Eh | “SQLFGLS” get long size |
| 2Fh | “SQLFELO” end long operation |
| 30h | “SQLFRBK” roll back a transaction from the database |
| 31h | “SQLFERR” error message |
| 32h | “SQLFCPY” copy |
| 33h | “SQLFR01” reserved |
| 34h | “SQLFSYS” system |
| 35h | “SQLFSTA” statistics |
| 36h | “SQLFR02” reserved |
| 37h | “SQLFXAD” extra add |
| 38h | “SQLFXCN” extra character to number |
| 39h | “SQLFXDA” extra date add |
| 3Ah | “SQLFXDP” extra date picture |
| 3Bh | “SQLFXDV” extra divide |
| 3Ch | “SQLFXML” extra multiply |
| 3Dh | “SQLFXNP” extra number picture |
| 3Eh | “SQLFXPD” extra picture date |
| 3Fh | “SQLFXSB” extra subtract |
| 40h | “SQLFINS” install database |
| 41h | “SQLFDIN” deinstall database |
| 42h | “SQLFDIR” directory of databases |
| 43h | “SQLFTIO” timeout |
| 44h | “SQLFFQN” get fully qualified column name |
| 45h | “SQLFEXP” explain execution plan |
| 46h | “SQLFFER” get full error |
| 47h | “SQLFBKP” begin online backup |
| 48h | “SQLFRDC” read backup data chunk |
| 49h | “SQLFEBK” end backup |
| 4Ah | “SQLFRES” begin restore from backup |
| 4Bh | “SQLFWDC” write backup data chunk for restore |
| 4Ch | “SQLFRRD” recover restored database to consistent state |
| 4Dh | “SQLFERS” end restore |
| 4Eh | “SQLFNRR” return number of result set rows |
| 4Fh | “SQLFSTR” start restriction mode |
| 50h | “SQLFSPR” stop restriction mode |
| 51h | “SQLFCNC” connect 2 |
| 52h | “SQLFCNR” connect with no recovery |
| 53h | “SQLFOMS” set output message size |
| 54h | “SQLFIMS” set input message size |
| 55h | “SQLFSCP” set cache pages |
| 56h | “SQLFDSC” describe items of a SELECT statement (external) |
| 57h | “SQLFLAB” get label info for items in SELECT statement |

| | |
|---|---|
| 58h | "SQLFCBV" clear bind variables |
| 59h | "SQLFGET" get database information |
| 5Ah | "SQLFSET" set database information |
| 5Bh | "SQLFTEC" translate error code |

---

## INT 47h Function 8001h

*SQL Base*

### *GET VERSION NUMBER*

**Registers at call:**
AX = 8001h

**Return Registers:**
*unknown*

**Conflicts:** Acorn BBC Master (Chapter 7), Z100 (Chapter 7), Western Digitial SuperBIOS (Chapter 10)

**See also:** Function 8000h

---

## INT 48h

*Compaq UILIB.EXE (undocumented)*

### *API*

**Registers at call:**
AX = function (see Table 54-3)
BX = call type (0002h) (see Table 54-4)
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** UILIB returns AX=FFFFh if AX is between 1000h and 2000h, inclusive, and AX is not one of the functions listed in Table 54-3.

**Conflicts:** Acorn BBC Master (Chapter 7), Z100 (Chapter 7), Western Digitial SuperBIOS (Chapter 10), PCjr (Chapter 13), Watstar PC Network (NI)

**See also:** Function 1A70h

**54-3 Values for function:**

1000h
1010h
1020h
1030h
1040h
1050h
1060h
1070h
1080h
1090h
1095h
1098h
10A0h
10C0h
10D0h
10E0h
10F0h
1100h
1110h
1120h
1130h
1140h
1150h
1160h
1170h
1180h
1190h
11A0h
11B0h
11C0h
11D0h
11E0h
11F0h
1200h
1210h
1220h
1230h
1240h
1250h
1260h
1270h
1280h
1290h
12A0h
12B0h
12C0h
12D0h
12E0h
12F0h
1300h
1310h
1320h
1330h
1340h
1350h
1360h
1370h
1380h
1390h
13A0h
13B0h
13B8h
13C0h
13D0h
13E0h
13F0h
1400h
1410h
1420h
1430h
1440h
1450h
1460h
1470h
1480h
1490h
14A0h
14B0h
14B8h
14BBh
14C0h
14D0h
14E0h
14F0h
1500h
1508h
1510h
1520h
1530h
1540h
1550h
1560h
1570h
1578h
1580h
1590h
1594h
1598h
15A0h
15B0h
15C0h
15D0h
15D4h
15D8h
15E0h
15F0h
1600h
1610h
1620h
1630h
1640h
1650h
1660h
1664h
1670h
1680h
1690h
16A0h
16B0h
16C0h
16D0h
16E0h
16F0h
1700h
1710h
1720h
1730h
1735h
1740h
1750h
1770h
1780h
1790h
17A0h
17B0h
17C0h
17D0h
17E0h
17F0h
1800h
1810h
1820h
1830h
1840h
1848h
1850h
1860h
1870h
1878h
1880h
1890h
1898h
18A0h
18B0h
18C0h
18D0h
18E0h
18F0h
1900h
1909h
1910h
1920h
1930h
1940h
1950h
1960h
1970h
1980h
1990h
19A0h
19B0h
19C0h
19D0h
19E0h
19F0h
1A00h
1A10h
1A20h
1A30h
1A40h
1A50h
1A60h
1A70h
1A80h
1A90h
1AA0h

---

### INT 48h Function 1A70h
### *INSTALLATION CHECK*

*Compaq UILIB.EXE (undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = 1A70h | CX = 5649h ('VI') if installed |
| BX = call type (see Table 54-4) | DX = 4557h ('EW') if installed |
| | AX = *version* (0106h) |

**Conflicts:** Acorn BBC Master (Chapter 7), Z100 (Chapter 7), Western Digitial SuperBIOS (Chapter 10), PCjr (Chapter 13), Watstar PC Network (NI)

**54-4 Values for call type:**

| | |
|---|---|
| 0000h | near |
| 0001h | far |
| 0002h | INT (only valid type when using INT 48h) |
| 0003h | near |

---

### INT 60h
### *PROCEDURE ENTRY TRAP*

*JPI TopSPEED Modula-2 v1*

**Conflicts:** see Table 2-3
**See also:** JPI INT 61h

---

### INT 61h
### *PROCEDURE EXIT TRAP*

*JPI TopSPEED Modula-2 v1*

**Conflicts:** see Table 2-3
**See also:** JPI INT 60h

---

### INT 62h Functions 00h to 07h
### *UNDETERMINED FUNCTION*

*MS SQL Server/Sybase DBLIBRARY interface*

| Registers at call: | Return Registers: |
|---|---|
| AH = function (00h to 07h) | *unknown* |
| CX = FFFEh | |
| DX = FFFFh | |
| *others, if any, unknown* | |

**Details:** The installation check consists of testing for the string "DBLIBRARY" 2 bytes past the interrupt handler.
**Conflicts:** see Table 2-3
**See also:** Function 08h

---

### INT 62h Function 08h
### *UNINSTALL/GET PSP ADDRESS*

*MS SQL Server/Sybase DBLIBRARY interface*

| Registers at call: | Return Registers: |
|---|---|
| AH = 08h | AX = PSP address of resident DBLIBRARY |
| CX = FFFEh | |
| DX = FFFFh | |

**Details:** This call does not free the memory allocated to the TSR; the calling code must do the deallocation.
**Conflicts:** see Table 2-3
**See also:** INT 62h

---

### INT 63h and INT 64h
### *UNDETERMINED FUNCTIONS*

*Oracle SQL Protected Mode Executive*

**Conflicts:** see Table 2-3

**INT 7Bh** *Btrieve (single user)*

***API***

**Registers at call:**
DS:DX -> 38-byte parameter record (see Table 54-5)

**Return Registers:**
return code field set

**Details:** Btrieve sets the low byte of this vector to 33h; this serves as the installation check.

**Conflicts:** GO32 (Chapter 37), AutoCAD (Chapter 66), Eicon Access (NI)

**See also:** INT 2Fh Function AB01h, Btrieve INT 7Fh Function 0200h

**54-5 Format of Btrieve parameter record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | pointer to data buffer |
| 04h | WORD | data buffer length |
| 06h | DWORD | pointer to 90-byte record containing positioning info (should be same for all calls for same file) |
| 0Ah | DWORD | pointer to 38-byte FCB info buffer (should be same for all calls for same file) |
| 0Eh | WORD | function code (see Table 54-6) |
| 10h | DWORD | pointer to file name/key buffer |
| 14h | BYTE | key length |
| 15h | BYTE | key number |
| 16h | DWORD | pointer to status code (see Table 54-7) |
| 1Ah | WORD | interface code (version specific); 6176h = version 5.10 |

**54-6 Values for function code:**

| | |
|---|---|
| 00h | open |
| 01h | close |
| 02h | insert |
| 03h | update |
| 04h | delete |
| 05h | get_equal |
| 06h | get_next |
| 07h | get_prev |
| 08h | get_greater |
| 09h | get_gr_eql |
| 0Ah | get_less |
| 0Bh | get_less_eq |
| 0Ch | get_first |
| 0Dh | get_last |
| 0Eh | create |
| 0Fh | stat |
| 10h | extend |
| 11h | set_dir: set directory information |
| 12h | get_dir: get directory information |
| 13h | begin_trans |
| 14h | end_trans |
| 15h | abort_trans |
| 16h | get_pos: get record position number |
| 17h | get_direct: get data by sending record position |
| 18h | step_next |
| 19h | stop |
| 1Ah | version |

| | |
|---|---|
| 1Bh | unlock |
| 1Ch | reset |
| 1Dh | set owner |
| 1Eh | clear owner |
| 1Fh | create supplemental index |
| 20h | drop supplemental index |
| 21h | step first |
| 22h | step last |
| 23h | step previous |
| 24h | get next extended: get multiple records using a filter |
| 25h | get previous extended: get multiple records using a filter |
| 26h | step next extended: get multiple records using a filter |
| 27h | step previous extended: get multiple records using a filter |
| 28h | insert extended: insert one or more records |
| 31h | *unknown* |

add 50 (32h) to any "get" operation to just return the key data

add 100 (64h) for a single-record wait lock (automatically released on next get)

add 200 (C8h) for a single-record nowait lock (nowait lock returns error 54h or 55h if record already locked)

add 300 (12Ch) for a multiple-record wait lock (not released until unlock called)

add 400 (190h) for a multiple-record nowait lock (nowait lock returns error 54h or 55h if record already locked)

**54-7 Values for status code:**

| | |
|---|---|
| 00h | successful |
| 01h | invalid operation |
| 02h | I/O error |
| 03h | file not open |
| 04h | key value not found |
| 05h | duplicate key value |
| 06h | invalid key number |
| 07h | different key number |
| 08h | invalid positioning |
| 09h | end of file |
| 0Ah | modifiable key value error |
| 0Bh | invalid file name |
| 0Ch | file not found |
| 0Dh | extended file error |
| 0Eh | pre-image open error |
| 0Fh | pre-image I/O error |
| 10h | expansion error |
| 11h | close error |
| 12h | disk full |
| 13h | unrecoverable error |
| 14h | record manager inactive |
| 15h | key buffer too short |
| 16h | data buffer length overrun |
| 17h | position block length |
| 18h | page size error |
| 19h | create I/O error |

| | |
|---|---|
| 1Ah | number of keys |
| 1Bh | invalid key position |
| 1Ch | invalid record length |
| 1Dh | invalid key length |
| 1Eh | not a Btrieve file |
| 1Fh | file already extended |
| 20h | extended I/O error |
| 22h | invalid extension name |
| 23h | directory error |
| 24h | transaction error |
| 25h | transaction is active |
| 26h | transaction control file I/O error |
| 27h | end/abort transaction error |
| 28h | transaction max files |
| 29h | operation not allowed |
| 2Ah | incomplete accelerated access |
| 2Bh | invalid record address |
| 2Ch | null key path |
| 2Dh | inconsistent key flags |
| 2Eh | access to file denied |
| 2Fh | maximum open files |
| 30h | invalid alternate sequence definition |
| 31h | key type error |
| 32h | owner already set |
| 33h | invalid owner |
| 34h | error writing cache |
| 35h | invalid interface |
| 36h | variable page error |
| 37h | autoincrement error |
| 38h | incomplete index |
| 39h | expanded memory error |
| 3Ah | compression buffer too short |
| 3Bh | file already exists |
| 3Ch | reject count reached |
| 3Dh | work space too small |
| 3Eh | incorrect descriptor |
| 3Fh | invalid extended insert |
| 40h | filter limit reached |
| 41h | incorrect field offset |
| 4Ah | automatic transaction abort |
| 4Eh | deadlock detected |
| 50h | conflict |
| 51h | lock error |
| 52h | lost position |
| 53h | read outside transaction |
| 54h | record in use |
| 55h | file in use |
| 56h | file table full |
| 57h | handle table full |

## INT 7Bh

| | |
|---|---|
| 58h | incompatible open mode |
| 5Ah | redirected device table full |
| 5Bh | server error |
| 5Ch | transaction table full |
| 5Dh | incompatible lock type |
| 5Eh | permission error |
| 5Fh | session no longer valid |
| 60h | communications environment error |
| 61h | data message too small |
| 62h | internal transaction error |

---

### INT 7Fh Function 0200h

*Btrieve Multi-User*

***GIVE UP TIME***

**Registers at call:**
AX = 0200h

**Return Registers:**
none

**Conflicts:** see Table 2-4
**See also:** INT 2Fh Functions AB01h and AB02h, Btrieve INT 7Bh

---

### INT 80h to INT 85h

*BASIC*

***Reserved for BASIC***

**Details:** These vectors and INT 86h through INT EDh are modified but not restored by Direct Access v4.0, and may be left dangling by other programs written with the same version of compiled BASIC.
**Conflicts:** SoundBlaster SBFM (Chapter 22) Q-PRO (Chapter 66), QPC PKTINT (NI)
**See also:** ROM BASIC INT 86h

---

### INT 86h to INT E3h

*IBM ROM BASIC*

***USED WHILE IN INTERPRETER***

**Details:** These vectors are called by IBM's ROM BASIC. BASIC.COM/BASICA.COM hook them at startup but do not restore them on termination.

INTs 86h, 88h, 8Bh-92h, 94h, 98h-9Eh, A1h, A4h, A5h, AAh, AEh, B0h, B1h, B3h, B5h, B7h, B9h-C4h, C6h-D3h, D5h-D7h, D9h, and DAh are pointed at IRETs by BASIC.COM and BASICA.COM.

INT 80h through INT EDh are modified but not restored by Direct Access v4.0, and may be left dangling by other programs written with the same version of compiled BASIC.
**Conflicts:** see Table 2-1
**See also:** BASIC INT 80h, BASIC INT E4h

---

### INT E4h Functions 0005h and 0006h

*Logitech Modula v2.0*

***MonitorEntry / MonitorExit***

**Registers at call:**
AX = function
    0005h MonitorEntry, 0006h MonitorExit
BX = priority

**Return Registers:**
none

**Conflicts:** ROM BASIC
**See also:** Function 0006h

---

### INT E4h to INT ECh

*IBM ROM BASIC*

***used while in interpreter***

**Details:** These vectors are called by IBM's ROM BASIC. BASIC.COM/BASICA.COM hook them at startup but do not restore them on termination.

INT E5h and INT E8h are pointed at IRETs by BASIC.COM and BASICA.COM.

INT 80h through INT EDh are modified but not restored by Direct Access v4.0, and may be left dangling by other programs written with the same version of compiled BASIC.

**Conflicts:** see Table 2-1

**See also:** BASIC INT 80h, BASIC INT 86h, BASIC INT EDh

---

## INT ECh

*Exact*

### *RUNTIME INTERFACE MULTIPLEXOR*

**Purpose:** This is the interface from applications to the runtime system by Exact Automatisering B.V. of the Netherlands. By using this interrupt, it can provide DLL-style capabilities under MS-DOS.

| Registers at call: | Return Registers: |
|---|---|
| AX = function number (0000h to 0140h) | STACK: return address popped, but otherwise unchanged |
| STACK: DWORD address to return to<br>any arguments required by function | |

**Details:** The interrupt handler removes the return address and flags placed on the stack by the INT ECh, then jumps to the appropriate function.

**Conflicts:** ROM BASIC, Alloy NTNX (NI)

---

## INT EDh and INT EEh

*IBM ROM BASIC*

### *USED WHILE IN INTERPRETER*

**Details:** These vectors are called by IBM's ROM BASIC. BASIC.COM/BASICA.COM hook them and point them at IRETs at startup but do not restore them on termination.

INT 80h through INT EDh are modified but not restored by Direct Access v4.0, and may be left dangling by other programs written with the same version of compiled BASIC.

**See also:** BASIC INT 80h, INT 86h, INT E4h, and INT EFh

---

## INT EFh

*BASIC*

### *ORIGINAL INT 09h VECTOR*

**Details:** BASIC.COM/BASICA.COM do not restore this vector on termination.

**Conflicts:** GEM

**See also:** BASIC INT EDh and INT F0h

---

## INT EFh

*GEM*

### *INTERFACE*

| Registers at call: | Return Registers: |
|---|---|
| CX = 0473h | none |
| DS:DX -> GEM parameter block | |

**Conflicts:** ROM BASIC

---

## INT F0h

*BASICA.COM, GWBASIC, compiled BASIC*

### *ORIGINAL INT 08h VECTOR*

**Details:** BASICA.COM does not restore this vector on termination.

**See also:** BASIC INT 86h, INT E4h, and INT EFh

## CHAPTER 55

# Intel Image Processing Interface

The Image Processing Interface permits communication with Intel's printer controller products such as the Visual Edge. The functions described in this chapter conflict with the LaserPort interface (Chapter 15) and SWELL.EXE (Chapter 49); as with all programs using multiplex numbers in the range C0h to FFh, there are numerous potential conflicts which are listed in Table 2-2.

---

### INT 2Fh Function CD00h
### *INSTALLATION CHECK*

*Intel Image Processing Interface*

**Registers at call:**
AX = CD00h

**Return Registers:**
AL = status
  00h not installed, OK to install
  01h not installed, not OK to install
  FFh installed

**See also:** Function CD02h

---

### INT 2Fh Function CD01h
### *SET DEVICE NAME*

*Intel Image Processing Interface*

**Registers at call:**
AX = CD01h
CX:BX -> ASCIZ character device name ("LPTn", "COMn", "PRN")

**Return Registers:**
AL = status
  00h successful
    CX:BX -> internal character device name
  80h error

---

### INT 2Fh Function CD02h
### *GET VERSION NUMBER*

*Intel Image Processing Interface*

**Registers at call:**
AX = CD02h

**Return Registers:**
AL = status
  00h/01h successful
    BH = major version number (BCD)
    BL = minor version number (BCD)
  80h error

**Details:** If return AL = 01h, the IPI supports network redirection.
**See also:** Function CD00h

---

### INT 2Fh Function CD03h
### *SELECT SCAN LINE*

*Intel Image Processing Interface*

**Registers at call:**
AX = CD03h
BX = scan line

CX = requested density in dots per inch (300, 600, or 1200)

**Return Registers:**
AL = status (00h,80h-83h) (see Table 55-1)
00h succesful
CX = density at which scan line was mapped
ES:DI -> start of scan line

**See also:** Function CD04h

**55-1 Values for status:**

| | |
|---|---|
| 00h | successful |
| 80h | unsuccessful |
| 81h | scan line out of range |
| 82h | unsupported scan line density |
| 83h | out of memory |
| 84h | unrecognized source |
| 85h | initialization error |

---

### INT 2Fh Function CD04h
*Intel Image Processing Interface*

### *MOVE BITMAP TO SCANLINE*

**Registers at call:**
AX = CD04h
CX:BX -> copy parameters (see Table 55-2)

**Return Registers:**
AL = status (00h,80h-85h) (see Table 55-1)

**See also:** Function CD03h

**55-2 Format of copy parameters:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | image source (0 = conventional memory, 1 = expanded memory) |
| 02h | DWORD | pointer to image data |
| 06h | WORD | scan line on which to place |
| 08h | WORD | bit offset from start of scan line at which to place |
| 0Ah | WORD | density of bitmap data (300, 600, or 1200 dpi) |
| 0Ch | WORD | width in bits of data |
| 0Eh | WORD | source logical page number |
| 10h | WORD | source handle (only if source in expanded memory) |
| 12h | WORD | source offset (only if source in expanded memory) |

---

### INT 2Fh Function CD05h
*Intel Image Processing Interface*

### *PRINT PAGE*

**Registers at call:**
AX = CD05h

**Return Registers:**
AL = status (00h,80h) (see Table 55-1)

**Details:** The page image is retained, so multiple calls will print multiple copies of the page.

**See also:** Function CD06h

---

### INT 2Fh Function CD06h
*Intel Image Processing Interface*

### *CLEAR PAGE*

**Registers at call:**
AX = CD06h

**Return Registers:**
AL = status (00h,80h) (see Table 55-1)

**Details:** The palette is reset to the default.

**See also:** Function CD09h

---

### INT 2Fh Function CD07h
*Intel Image Processing Interface*

### *RESERVED*

**Registers at call:**
AX = CD07h

**Return Registers:**
none

## INT 2Fh Function CD08h

*Intel Image Processing Interface*

### *SCREEN IMAGE*

**Registers at call:**
AX = CD08h
CX:BX -> screening parameters (see Table 55-3)

**Return Registers:**
AL = status (00h,80h-85h)
(see Table 55-1)

**See also:** Functions CD05h and CD09h

**55-3 Format of screening parameters:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | image source (0 = conventional memory, 1 = expanded memory) |
| 02h | DWORD | pointer to image data |
| 06h | WORD | horizontal position on paper of left edge (in 1200 dpi units) |
| 08h | WORD | vertical position on paper of top edge (in 1200 dpi units) |
| 0Ah | WORD | left cropping (currently must be zero) |
| 0Ch | WORD | top cropping (currently must be zero) |
| 0Eh | WORD | width (currently must be 8000h) |
| 10h | WORD | height (currently must be 8000h) |
| 12h | WORD | horizontal size of image in 1200 dpi units |
| 14h | WORD | vertical size of image in 1200 dpi units |
| 16h | WORD | aspect ratio (currently reserved) |
| 18h | WORD | initialization flag (if 01h, initialization is performed) |
| 1Ah | WORD | pixels per line of source data |
| 1Ch | WORD | number of scan lines in source data |
| 1Eh | WORD | number of scan lines in packet |
| 20h | WORD | bits per pixel (1,2,4,6, or 8) |
| 22h | WORD | pixels per byte (1,2,4, or 8) |
| 24h | WORD | compression type (currently only 00h [uncompressed] supported) |
| 26h | WORD | source page number (if in expanded memory) |
| 28h | WORD | source handle (if in expanded memory) |
| 2Ah | WORD | source offset (if in expanded memory) |

## INT 2Fh Function CD09h

*Intel Image Processing Interface*

### *LOAD SCREEN*

**Registers at call:**
AX = CD09h
CX:BX -> load parameters (see Table 55-4)

**Return Registers:**
AL = status (00h,80h) (see Table 55-1)

**See also:** Functions CD06h and CD0Ah

**55-4 Format of load parameters:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | style<br>44h ('D') diamond style<br>4Ch ('L') line style |
| 01h | BYTE | reserved (00h) |
| 02h | WORD | frequency in lines per inch [sic]<br>currently, coerced to nearest of 50, 60, 68, 70, 75, 85, or 100 |
| 04h | WORD | screen angle in degrees (-360 to 360)<br>currently coerced to nearest of -45, 0, 45, or 90 |

## INT 2Fh Function CD0Ah

Intel Image Processing Interface

### *LOAD PALETTE*

**Registers at call:**
AX = CD0Ah
CX:BX -> palette structure (see Table 55-5)

**Return Registers:**
AL = status (00h,80h) (see Table 55-1)

**See also:** Function CD09h

**55-5 Format of palette structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | bits per pixel for which palette is to be used (1,2,4,6, or 8) |
| 01h | 2**N | palette translation values, one per possible pixel value |

CHAPTER 56

# System Profilers

A profiler is a tool intended to help you locate which functions are being used within a program and moreover, to show how the program's execution time is split up among the functions. A number of profilers are available and many of them use interrupts to communicate between the profiler itself and the program being profiled.

## Microsoft Profiler

The Microsoft profiler (really a collection of three separate programs which prepare, profile, and report) is designed to be able to help you determine completeness of code coverage during program testing.

---

**INT 2Fh Function 4500h** *Microsoft Profiler (PROF.COM/VPROD.386)*
***INSTALLATION CHECK*** *(undocumented)*

**Registers at call:**
AX = 4500h

**Return Registers:**
AL = 01h if PROF.COM installed
AL = 02h if VPROD.386 installed

**See also:** Functions 4501h and 4502h

---

**INT 2Fh Function 4501h** *Microsoft Profiler (PROF.COM/VPROD.386)*
***SETUP PROFILER*** *(undocumented)*

**Registers at call:**
AX = 4501h
BX = CSIPS buffer size in KB (first parameter for ProfSetup)
CX = output limit in KB (second parameter for ProfSetup)

**Return Registers:**
none

**Details:** This call is not supported by PROF.COM.
**See also:** Functions 4502h and 4503h

---

**INT 2Fh Function 4502h** *Microsoft Profiler (PROF.COM/VPROD.386)*
***SET SAMPLING RATE*** *(undocumented)*

**Registers at call:**
AX = 4502h
BL = sampling rate for PROF.COM (01h-0Dh)
(01h=8192/s, 04h=1024/s, 08h=32/s, 0Dh=1/s)
CX = sampling rate for VPROD.386

**Return Registers:**
none

**Details:** For PROF.COM, this function programs the CMOS clock by setting BL+2 as the low four bits of CMOS register 0Ah. The interruption rate is 1 SHL (15 - BL) per second.
**See also:** Functions 4501h and 4503h

---

**INT 2Fh Function 4503h** *Microsoft Profiler (PROF.COM/VPROD.386) (undocumented)*

***START PROFILING***

**Registers at call:** AX = 4503h

**Return Registers:** none

**Details:** Profiling is also turned on by the key combinations LeftShift + RightShift + Alt LeftShift + RightShift + Ctrl.

For PROF.COM, this call programs the CMOS clock by reading register 0Ch, and setting bit 6 of register 0Bh. It then makes sure that IRQ8 is unmasked.

**See also:** Function 4504h

---

**INT 2Fh Function 4504h** *Microsoft Profiler (PROF.COM/VPROD.386) (undocumented)*

***STOP PROFILING***

**Registers at call:** AX = 4504h

**Return Registers:** none

**Details:** Profiling is also turned off by the key combination LeftShift + RightShift.

For PROF.COM, this programs the CMOS clock by reading register 0Ch and clearing bit 6 of register 0Bh. It then masks IRQ8.

**See also:** Functions 4503h, 4505h, 4506h, and 4507h

---

**INT 2Fh Function 4505h** *Microsoft Profiler (PROF.COM/VPROD.386) (undocumented)*

***CLEAR PROFILING DATA***

**Registers at call:** AX = 4505h

**Return Registers:** none

**See also:** Functions 4503h, 4504h, and 4506h

---

**INT 2Fh Function 4506h** *Microsoft Profiler (PROF.COM/VPROD.386) (undocumented)*

***"ProfFlush"***

**Registers at call:** AX = 4506h

**Return Registers:** none

**See also:** Functions 4505h and 4507h

---

**INT 2Fh Function 4507h** *Microsoft Profiler (PROF.COM/VPROD.386) (undocumented)*

***"ProfFinish"***

**Registers at call:** AX = 4507h

**Return Registers:** none

**Details:** This call is essentially a "ProfStop" (Function 4504h) followed by "ProfFlush" (Function 4506h).

**See also:** Functions 4504h, 4505h, and 4506h

---

**INT 2Fh Function 4508h** *Microsoft Profiler (PROF.COM/VPROD.386) (undocumented)*

***ALTERNATE SEGDEBUG IFACE***

**Registers at call:**
AX = 4508h
BX = ordinal (or 0000h)
CX = segment
DX = instance (or 0000h)
SI = type (or 0000h)
ES:DI -> ASCIZ module name

**Return Registers:** none

**Details:** This call is an alternate entry to the profiler's SEGDEBUG interface, but only to function 0, for notifying the profiler of each new segment loaded. The SHOWHITS utility then examines the profiler's output files (CSIPS.DAT and SEGENTRY.DAT) in conjunction with symbol files to provide information in a useful form. This call does not have a corresponding Windows function.

**See also:** Function 4500h

## SYS_PROF

SYS_PROF.EXE is the TSR portion of a profiler from Micro Cornucopia magazine, Issue #47. As with all programs using interrupts in the range 60h to 67h, there are numerous potential conflicts, which are listed in Table 2-3.

---

### INT 60h Function 00h
*SYS_PROF.EXE*

***PROFILER STATUS***

**Registers at call:**
AH = 00h

**Return Registers:**
AX = profiling status (0000h off, else on)

**See also:** Functions 01h and 02h

---

### INT 60h Function 01h
*SYS_PROF.EXE*

***TURN PROFILING OFF***

**Registers at call:**
AH = 01h

**Return Registers:**
none

**See also:** Functions 00h and 02h

---

### INT 60h Function 02h
*SYS_PROF.EXE*

***TURN PROFILING ON***

**Registers at call:**
AH = 02h

**Return Registers:**
none

**See also:** Functions 00h and 01h

---

### INT 60h Function 03h
*SYS_PROF.EXE*

***GET ADDRESS OF PROFILING TABLE***

**Registers at call:**
AH = 03h

**Return Registers:**
ES:BX -> profiling table

**See also:** Function 04h

---

### INT 60h Function 04h
*SYS_PROF.EXE*

***CLEAR PROFILING TABLE***

**Registers at call:**
AH = 04h

**Return Registers:**
none

**See also:** Function 03h

## CHAPTER 57

# *Debuggers and Debugging Tools*

Debugging tools run the gamut from the simple instruction interceptor typified by DEBUG.EXE through complex systems such as INTRSPY that install hooks on specified interrupts and create formatted reports of a program's entire history. This chapter lists interrupts used by five major debugging tools together with miscellaneous entries.

## CAPDOS

CAPDOS is a TSR by Nanosoft, Inc. which allows INT 21h calls to be captured and recorded for later analysis.

---

### INT 2Fh Function 8200h

***INSTALLATION CHECK***

*CAPDOS*
*(undocumented)*

**Registers at call:**
AX = 8200h

**Return Registers:**
AL = FFh if installed

**See also:** Functions 8201h, 8202h, 8203h, and 8204h; TurboNET Function 8100h (NI)

---

### INT 2Fh Function 8201h

***CLEAR QUEUE***

*CAPDOS*

**Registers at call:**
AX = 8201h

**Return Registers:**
none

**Details:** This function resets the queue of captured INT 21h calls.

**See also:** Function 8200h

---

### INT 2Fh Functions 8202h and 8203h

***START/STOP COLLECTION***

*CAPDOS*

**Registers at call:**
AX = function (8202h start, 8203h stop)

**Return Registers:**
none

**See also:** Functions 8200h and 8204h

---

### INT 2Fh Function 8204h

***GET QUEUE PARAMETERS***

*CAPDOS*

**Registers at call:**
AX = 8204h

**Return Registers:**
AH = flag: queue wrapped if nonzero
BX = index of current start of queue
CX = size of queue in entries

**See also:** Functions 8200h, 8202h, and 8205h

---

**INT 2Fh Function 8205h** *CAPDOS*

***GET QUEUE ITEM***

**Registers at call:**
AX = 8205h
BX = queue item number

**Return Registers:**
AX,BX,CX,DX,SI,DI,DS,ES as on entry to captured DOS call

**See also:** Functions 8200h and 8204h

## COMTROL Hostess i/ISA Debugger

Comtrol is one of the original suppliers of multiport serial cards for the IBM-PC (ISA) architecture.

---

**INT 20h** *COMTROL HOSTESS i/ISA DEBUGGER*

***INVOKE FIRMWARE DEBUGGER***

**Registers at call:**
*unknown*

**Return Registers:**
*unknown*

**Conflicts:** MS-DOS (Chapter 24)
**See also:** INT 21h

---

**INT 21h** *COMTROL HOSTESS i/ISA DEBUGGER*

***GET SEGMENT FOR CONTROL PROGRAM USE***

**Registers at call:**
*unknown*

**Return Registers:**
AX = first segment available for control program use

**Conflicts:** MS-DOS (Chapter 24)
**See also:** INT 20h, INT 22h

---

**INT 22h** *COMTROL HOSTESS i/ISA DEBUGGER*

***CHANGE FIRMWARE DEBUGGING PORT***

**Registers at call:**
AL = new firmware debugging port

**Return Registers:**
*unknown*

**Conflicts:** MS-DOS (Chapter 24)
**See also:** INT 21h, INT 23h

---

**INT 23h** *COMTROL HOSTESS i/ISA DEBUGGER*

***GET CONFIGURATION INFORMATION***

| Registers at call: | Return Registers: |
|---|---|
| AL = query type | |
| 00h get old config map | AX = old config map |
| 01h get dual-ported RAM map | BX:AX = dual-ported RAM map |
| 02h get SCC port map | BX:AX = SCC port map |

**Conflicts:** MS-DOS (Chapter 24)
**See also:** INT 22h, INT 26h

---

**INT 26h** *COMTROL HOSTESS i/ISA DEBUGGER*

***ENTER/EXIT EXTENDED ADDRESSING MODE***

**Registers at call:**
*unknown*

**Return Registers:**
*unknown*

**Conflicts:** MS-DOS (Chapter 24)
**See also:** INT 23h, INT 27h

---

**INT 27h** *COMTROL HOSTESS i/ISA DEBUGGER*

***INVOKE REMOTE TURBO DEBUGGER KERNEL***

**Purpose:** Invoke a copy of the remote Turbo Debugger kernel on the Hostess i controller.

**Registers at call:** *unknown*

**Return Registers:** *unknown*

**Conflicts:** MS-DOS (Chapter 24)

**See also:** INT 20h, INT 26h

## INTMON

INTMON is a shareware interactive interrupt monitoring TSR for 386 and higher machines by Celso Minnitti, Jr.; INTCFG is an optionally-resident control program for INTMON.

---

### INT 2Fh Function 9E00h
***INSTALLATION CHECK***

*INTMON v2.1*
*(undocumented)*

**Registers at call:**
AX = 9E00h

**Return Registers:**
AX = FFFFh if installed
BX = segment of resident code

**See also:** Functions 9E01h, 9E02h, 9E03h, and 9F00h

---

### INT 2Fh Function 9E01h
***RESET***

*INTMON v2.1*
*(undocumented)*

**Purpose:** This function specifies that INTMON should assume that any interrupts on which it is currently awaiting a return have completed (i.e. interrupts which never return such as INT 20h and INT 27h).

**Registers at call:**
AX = 9E01h

**Return Registers:**
*unknown*

**See also:** Functions 9E00h and 9E03h

---

### INT 2Fh Function 9E02h
*DISPLAY CPU REGISTERS*

*INTMON v2.1*
*(undocumented)*

**Registers at call:**
AX = 9E02h

**Return Registers:**
*unknown*

**See also:** Function 9E00h

---

### INT 2Fh Function 9E03h
*HOOK INTERRUPT*

*INTMON v2.1*
*(undocumented)*

**Registers at call:**
AX = 9E03h
BH = *interrupt number*

**Return Registers:**
*unknown*

**Details:** If AH > 03h on entry, INTMON 2.1 returns immediately.

**See also:** Functions 9E00h and 9E01h

---

### INT 2Fh Function 9F00h
***INSTALLATION CHECK***

*INTCFG v2.1*
*(undocumented)*

**Registers at call:**
AX = 9F00h

**Return Registers:**
AX = FFFFh if installed

**See also:** Functions 9E00h, 9F01h, 9F30h, and 9F49h

---

### INT 2Fh Function 9F01h
*UNDETERMINED FUNCTION*

*INTCFG v2.1*
*(undocumented)*

**Registers at call:**
AX = 9F01h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 9F00h

---

**INT 2Fh Function 9F30h**
*INTCFG v2.1*
***GET** UNKNOWN VALUE*
*(undocumented)*

**Registers at call:**
AX = 9F30h

**Return Registers:**
AX = *unknown* (0002h)

**See also:** Function 9F00h

---

**INT 2Fh Function 9F49h**
*INTCFG v2.1*
***UNINSTALL***
*(undocumented)*

**Registers at call:**
AX = 9F49h

**Return Registers:**
AX,DX,DS,ES destroyed

**See also:** Function 9F00h

## MDEBUG

MDEBUG is a shareware memory-resident debugging tool by Bernd Schemmer, including a memory monitor, an interpreter and a disassembler.

MDEBUG can use any two consecutive multiplex numbers between C0h and FFh; the default is D0h for the display driver and D1h for the command driver (see INT 60h Function 00h for the actual multiplex numbers used). There are numerous potential conflicts on both INT 2Fh and INT 60h, which are listed in Tables 2-2 and 2.3, respectively.

---

**INT 2Fh Function D000h**
*MDEBUG display driver*
***GET DRIVER STATUS***
*Callout*

**Registers at call:**
AX = D000h

**Return Registers:**
CF set on error
    all other registers must be unchanged)
CF clear if successful
    AL = FFh
    AH = driver semaphor
        00h driver is not active
        01h driver is active
    BX = CS of the driver
    CX = driver version (CH = major, CL = minor, must be >= 013Ch)
    DL = buffer semaphor
        00h driver is not pending
        01h pending between functions 02h and 03h
    DH = show semaphor
        00h driver is not pending
        01h pending between functions 04h and 05h

**Details:** This function MUST be reentrant, as MDEBUG calls it after every popup before any other actions. The handler should not change any registers if the display is in an unsupported mode or in a mode MDEBUG supports itself, e.g. a normal text mode with at least 80x25 characters (i.e. 80x43 or 132x44 (v1.60+)). In this case MDEBUG will not call any of the other functions for this popup session.

MDEBUG will not call the other functions if the returned version is less than the actual version of MDEBUG.

If the driver is reentrant, DL and DH should be 00h.

**Conflicts:** ZWmous (Chapter 21), Lotus CD/Networker (NI); see also Table 2-2

**See also:** Functions D001h, D002h, D003h, D004h, and D005h

---

**INT 2Fh Function D001h**
*MDEBUG display driver*
***INITIALIZE DRIVER***
*Callout*

**Registers at call:**
AX = D001h

**Return Registers:**
CF set on error

AL = driver semaphor
AH = buffer semaphor

**Details:** MDEBUG calls this function after every succesful call of Function 00h. The handler should reset all internal data and the status of the driver. If this function returns an error, MDEBUG will not call the other functions in this popup session.

**See also:** Function D000h

---

### INT 2Fh Function D002h
### *SAVE GRAPHIC DATA*

*MDEBUG display driver*
*Callout*

**Registers at call:**
AX = D002h

**Return Registers:**
CF set on error
CF clear if successful
display memory saved and display switched to one of the text modes 02h, 03h or 07h

**Details:** MDEBUG calls this function only once every popup session before displaying its windows.

**See also:** Functions D000h and D003h

---

### INT 2Fh Function D003h
### *RESTORE GRAPHIC DATA*

*MDEBUG display driver*
*Callout*

**Registers at call:**
AX = D003h

**Return Registers:**
CF set on error
CF clear if successful
display restored to the mode it was in before calling Function D002h, and the display memory is restored

**Details:** MDEBUG calls this function only once every popup session just before it exits to normal DOS.

**See also:** Functions D000h and D002h

---

### INT 2Fh Function D004h
### *SHOW SAVED DATA*

*MDEBUG display driver*

**Registers at call:**
AX = D004h

**Return Registers:**
CF set on error
CF clear if successful
display switched to mode it was in before calling Function D002h, and the display memory is restored

**Details:** This function needn't save the display memory before changing it.

**See also:** Functions D000h, D002h, and D005h

---

### INT 2Fh Function D005h
### *SWITCH BACK TO TEXT SCREEN*

*MDEBUG display driver*

**Registers at call:**
AX = D005h

**Return Registers:**
CF set on error
CF clear if successful
display restored to mode it was in before calling Function D004h

**Details:** This function needn't save or change the display memory.

**See also:** Functions D000h and D004h

---

### INT 2Fh Function D0h
### *RESERVED FUNCTION NUMBERS*

*MDEBUG display driver*

**Registers at call:**
AH = D0h
AL = 06h-7Fh

**Return Registers:**
none

**Details:** These functions are reserved for future use.

### INT 2Fh Function D0h

*MDEBUG display driver*

#### *USER DEFINED FUNCTION NUMBERS*

**Registers at call:**
AH = D0h
AL = 80h-FFh

**Return Registers:**
vary by function and driver

**Details:** These function numbers are reserved for user defined features (e.g. communication between the transient and resident parts of the driver).

### INT 2Fh Function D100h

*MDEBUG command driver*
*Callout*

#### *GET STATUS*

**Registers at call:**
AX = D100h
BX = version of MDEBUG (BH = major, BL = minor)
CX = command driver counter
---v1.60+---
DS:SI -> MDEBUG identification table (see Table 57-1)
ES = segment of display memory used by MDEBUG
DI = size of video mode used by MDEBUG (high byte = lines, low byte = columns)

**Return Registers:**
DL = FFh
BX = version number of the driver if it is less than the version in BX, else unchanged
CX incremented

**Details:** This function is called by MDEBUG at the start of a popup session; if the version number returned in BX is less than 1.52 (1.60 for MDEBUG v1.70), MDEBUG will not call any of the other functions during this popup session.

This function must end with a far call to the old INT 2Fh handler after changing the registers.

This function MUST be reentrant.

Command drivers must also declare the following data at the given offsets in the code segment:

| | | |
|---|---|---|
| 100h | 3 BYTEs | JMP-command in .COM-files |
| 103h | BYTE | NOP-command (90h) |
| 104h | 26 BYTEs | signature "Kommandotreiber für MDEBUG" |
| 11Eh | 12 BYTEs | name of driver, e.g. "MDHISDRV.COM"; each driver must have a unique na |

MDEBUG will pass every key and command to the command driver(s) before checking for a valid internal command.

**See also:** Functions D000h and D101h

**57-1 Format of MDEBUG identification table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -2 | WORD | entry offset |
| 00h | WORD | CS of MDEBUG |
| 02h | DWORD | old INT 08h vector |
| 06h | DWORD | old INT 09h vector |
| 0Ah | DWORD | address INT 16h routine used by MDEBUG |
| 0Eh | BYTE | length of version string |
| 0Fh | N BYTEs | version string |

### INT 2Fh Function D101h

*MDEBUG command driver*

#### *INITIALIZE DRIVER*

**Registers at call:**
AX = D101h
CX = command driver counter

**Return Registers:**
DL = FFh if successful
CX incremented
else error: all registers unchanged

**Details:** This function must end with a far call to the old INT 2Fh handler after changing the registers, and it must be reentrant.

### INT 2Fh Function D102h

*MDEBUG command driver*

### *EXECUTE INTERPRETER COMMAND*

**Registers at call:**

AX = D102h
BL = first character of the interpreter command
BH = last character of the interpreter command (or blank)
DS:SI -> parameter for the interpreter command as ASCIZ string
DS:DI -> MDEBUG data structure (see Table 57-3)

**Return Registers:**

AL = FFh
CF set on error
- AH = error number (see Table 57-2)
  - DS:SI -> ASCIZ error message (max 30 characters)
- else unknown error

CF clear if successful
- AH = return code
  - 00h continue processing the command line
  - 01h leave MDEBUG popup session
  - 02h leave MDEBUG popup session and automatically popup again if the InDOS flag is zero
  - 03h not used (same as 00h)
  - 04h not used (same as 00h)
  - 05h put new command line into the input buffer,
    - DS:SI -> new command line (ASCIZ string, max 66 chars)
  - 06h process new command line
    - DS:SI -> new command line (ASCIZ string, max 66 chars)
- else unknown status, but continue processing commmand line

**Details:** This function must end with a far call to the old INT 2Fh handler (with registers unchanged) if the driver does not support the interpreter command in BX. Otherwise, the driver must not chain to the old INT 2Fh.

**57-2 MDEBUG error numbers:**

| | |
|---|---|
| 01h | syntax error |
| 02h | first shell of the command.com is activ |
| 03h | Esc pressed |
| 04h | break pressed |
| 05h | DOS is busy |
| 06h | command ended |
| 07h | division by zero |
| 08h | invalid display driver |
| 09h | invalid command driver |
| 0Ah | error 8 and 9 |
| 0Bh | unknown error |
| 0Ch | new error |

**57-3 Format of MDEBUG data structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 4 WORDs | registers SE, OF, FS, FO |
| 08h | 6 WORDs | registers AX, BX, CX, DX, SI, DI |
| 14h | 5 WORDs | registers DS, ES, BP, SS, SP |
| 1Eh | WORD | register FL (flags) |
| 20h | 4 WORDs | registers R0, R1, R2, R3 |
| 28h | 5 WORDs | registers R4, R5, R6, R7, R8 |
| 32h | WORD | register CS, return-address |
| 34h | WORD | register IP, return-address |

| | | |
|---|---|---|
| 36h | DWORD | saved pointer to data for key <F6> (v1.60) saved monitor address (v1.70) |
| 3Ah | 12 WORDs | saved register values on last popup entry (for <F8> key) (original register values at popup entry of MDEBUG) AX, BX, CX, DX, SI, DI, DS, ES, BP, SS, SP, flags |
| 52h | 12 WORDs | saved register values on last popup exit (for <SHIFT-F8> key) AX, BX, CX, DX, SI, DI, DS, ES, BP, SS, SP, flags |
| 6Ah | DWORD | address of the DOS-invars-table |
| 6Eh | DWORD | address of the InDOS flag |
| 72h | WORD | offset of the register which is used for the segment of the first monitor window |
| 74h | WORD | offset of the register which is used for the offset of the first monitor window |
| 76h | WORD | name of the register which is used for the segment of the first monitor segment |
| 78h | WORD | name of the register which is used for the offset of the first monitor window |
| 7Ah | WORD | pseudo register 1 |
| 7Ch | WORD | pseudo register 2 |

---

### INT 2Fh Function D103h

*MDEBUG command driver*

***EXECUTE KEY IN THE MONITOR***

**Registers at call:**
AX = D103h
BX = key code (like result of an interrupt 16h call)
CX = cursor position in monitor (0 in ASCII column, 1 in hex field)
DS:SI -> MDEBUG data structure (see Table 57-3)
ES:DI -> actual byte in the monitor

**Return Registers:**
AL = FFh
AH = return code
  00h key processed, read next key
  01h leave MDEBUG popup session
  02h leave MDEBUG popup session and automatically popup again if DOS is not busy
  03h signal an error (beep)
  04h driver has redefined the key, proceed with the new key
    BX = new key code
    MDEBUG will not pass the new key to the command driver(s)
  else treat like code 00h

**Details:** This function must end with a far call to the old INT 2Fh handler (with registers unchanged) if the driver does not support the key in BX. Otherwise, the driver must not chain to the old INT 2Fh.

**See also:** Function D104h

---

### INT 2Fh Function D104h

*MDEBUG command driver*

***EXECUTE KEY IN THE INTERPRETER***

**Registers at call:**
AX = D104h
DS:SI -> MDEBUG data structure (see Table 57-3)

**Return Registers:**
AL = FFh
AH = return code (see also Function D103h)
  05h put new command line into the input buffer
    DS:SI -> new command line (ASCIZ string, max 66 chars)
  06h process new command line
    DS:SI -> new command line (ASCIZ string, max 66 chars)
  else treat like code 00h

**Details:** (see Function D103h)

**See also:** Function D103h

### INT 2Fh Function D1h
*MDEBUG command driver*

### *RESERVED FUNCTIONS*

**Registers at call:**
AH = D1h
AL = 05h-0Ah

**Return Registers:**
none

**Details:** These functions are reserved for future use.

### INT 2Fh Function D110h
*MDEBUG command driver*

### *GET ADDRESS OF THE OLD INT 2Fh*

**Registers at call:**
AX = D110h

**Return Registers:**
DL = FFh
ES:BX -> next program in the chain for INT 2Fh
CX = code segment of this driver
DI = offset of driver identification table (see Table 57-4) (v1.60+)

**Details:** This function is called only by the transient part of the driver.

It must be reentrant and the driver must not chain this function to the old INT 2Fh.

**57-4 Format of the driver identification table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 26 BYTEs | signature “Kommandotreiber für MDEBUG” |
| 27h | 12 BYTEs | name of driver, e.g. “MDHISDRV.COM”; each driver must have a unique name |

### INT 2Fh Functions D111h and D112h
*MDEBUG command driver*

### *START/END DRIVER*

**Registers at call:**
AX = function
D111h start driver
D112h end driver

**Return Registers:**
DL = FFh

**Details:** These functions are called only by the transient part of the driver; Function D111h to inform the resident part that it is installed, and Function D112h to inform it that it is about to be released.

These functions must be reentrant and the driver must not chain them to the old INT 2Fh.

**See also:** Function D101h

### INT 2Fh Function D1h
*MDEBUG command driver*

### *RESERVED FUNCTIONS*

**Registers at call:**
AH = D1h
AL = 13h-7Fh

**Return Registers:**
none

**Details:** These functions are reserved for future use.

### INT 2Fh Function D1h
*MDEBUG command driver*

### *USER DEFINED FUNCTIONS*

**Registers at call:**
AH = D1h
AL = 80h-FFh

**Return Registers:**
vary by driver and function

**Details:** These functions are reserved for user defined features (e.g. communication between the transient and resident parts of the driver).

### INT 60h Function 00h
*MDEBUG*

### *GET STATUS*

**Registers at call:**
AH = 00h
DS:SI -> password or a null byte

**Return Registers:**
AX = return code
FFFEh password is invalid

FFFDh display mode is invalid
else successful
ES = value of the monitor register SE
DI = value of the monitor register OF
CH = monitor color
CL = interpreter color
BH = monitor start line
BL = interpreter start line
AH = makecode of the hotkey
AL = ASCII code of the hotkey
DL = status of special keys (only SHIFT, ALT, CTRL) for the hotkey (coded as for the keyboard flag at 0040h:0017h)
DH = basic process number for the communication with drivers process number for the display driver
DH+1 = process number for the command driver(s)
DS:SI -> MDEBUG identification table (see Table 57-5)

**Details:** MDEBUG uses INT 60h by default, but may be directed to any of USERINT INT 60h through INT 67h; the interrupt handler is preceded by the signature and is not chained.

If DS:SI points at a null byte, MDEBUG will prompt for a password if passwords are active; enough stack space must be provided for an INT 10h call (which MDEBUG uses while prompting for the password).

**Conflicts:** see Table 2-3

**See also:** Function 02h

**57-5 Format of MDEBUG identification table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -2 | WORD | entry offset |
| 00h | WORD | CS of MDEBUG |
| 02h | DWORD | old INT 08h vector |
| 06h | DWORD | old INT 09h vector |
| 0Ah | DWORD | address INT 16h routine used by MDEBUG |
| 0Eh | BYTE | length of version string |
| 0Fh | N BYTEs | version string |

## INT 60h Function 01h

*MDEBUG*

### *GET ADDRESS OF THE HELP REGISTERS*

**Registers at call:**
AH = 01h
DS:SI -> password or a null byte

**Return Registers:**
AX = return code
FFFEh password is invalid
FFFDh display mode is invalid else successful
ES:DI points to the MDEBUG help registers
ES:DI-02h -> R0 (WORD)
ES:DI -> R1 (WORD)
ES:DI+02h -> R2 (WORD)
ES:DI+04h -> R3 (WORD)
...
ES:DI+0Eh -> R8 (WORD)

## INT 60h Function 02h

*MDEBUG*

### *SET STATUS*

**Registers at call:**
AH = 02h

DS:SI -> password or a null byte
ES = new value for the register SE
DI = new value for the register OF
CH = new monitor color if nonzero
CL = new interpreter color if nonzero
BH = new monitor start line if nonzero
BL = new interpreter start line if nonzero
AL = new ASCII code for the hotkey ('A'..'Z', 'a'..'z') if nonzero
DL = new status of the special keys (SHIFT, ALT, CTRL) for the hotkey if nonzero
DH = if nonzero, new basic process number for communication with the drivers (DH = multiplex number for the display driver, DH+1 = multiplex number for the command driver or drivers)

**Return Registers:**
AX = return code
FFFFh call not allowed
FFFEh password is invalid
FFFDh display mode is invalid
0000h successful, status changed
else AL = error reasons (see Table 57-6)

**Details:** The values of the registers SE and OF are always changed, the other values are only changed if they are valid.
**See also:** Function 00h

**57-6 Bitfields for error reasons:**

| | |
|---|---|
| bit 0 | invalid monitor start line |
| bit 1 | invalid interpreter start line |
| bit 2 | invalid hotkey |
| bit 3 | invalid process number |
| bits 4-7 reserved | |

### INT 60h Function 03h

*MDEBUG*

***POP UP***

**Registers at call:**
AH = 03h
DS:SI -> password or a null byte
ES -> new value for the register SE
DI -> new value for the register OF

**Return Registers:**
AX = return code
FFFFh call not allowed
FFFEh password is invalid
FFFDh display mode is invalid
else successful

**See also:** Function 04h

### INT 60h Function 04h

*MDEBUG*

***POP UP***

**Registers at call:**
AH = 04h
DS:SI -> password or a null byte

**Return Registers:**
AX = return code (see Function 03h)

**Conflicts:** see Table 2-3
**See also:** Functions 03h and 07h

### INT 60h Function 05h

*MDEBUG*

***GET AND SET MDEBUG FLAGS***

**Registers at call:**
AH = 05h
DS:SI -> password or a null byte
BL = new value for the MDEBUG semaphor
    00h enable popup
    else disable popup

**Return Registers:**
AX = return code
FFFEh password is invalid
FFFDh display mode is invalid
else successful
    BL = old value of the MDEBUG semaphor
    BH = old value of the INT 08h semaphor (this semaphor is always reset after this function)

### INT 60h Function 06h
*MDEBUG*
### *GET PASSWORD STATUS*

**Registers at call:**
AH = 06h

**Return Registers:**
AL = password status
(00h inactive, 01h active)

### INT 60h Function 07h
*MDEBUG v1.70+*
### *GET ACTIVE PART OF MDEBUG*

**Registers at call:**
AH = 07h

**Return Registers:**
AL = active part for the next popup session of MDEBUG
bit 0: the next popup session will start in the interpreter rather than in the monitor
bit 1: the next popup session will sart in the online-help

**See also:** Functions 03h and 04h

### INT 60h Function 08h
*MDEBUG*
### *UNUSED*

**Registers at call:**
AH = 08h-FFh

**Return Registers:**
AX = FFFCh

## Soft-ICE

Soft-ICE is a debugger by Nu-Mega Technologies, Inc.

### INT 03h Function 0910h
*Soft-ICE*
### *BACK DOOR COMMANDS - DISPLAY STRING IN Soft-ICE WINDOW*

**Registers at call:**
AX = 0910h
SI = 4647h ('FG')
DI = 4A4Dh ('JM')
DS:DX -> ASCIZ string to display (max 100 bytes, 0Dh OK)

**Return Registers:**
none

### INT 03h Function 0911h
*Soft-ICE*
### *BACK DOOR COMMANDS - EXECUTE Soft-ICE COMMAND*

**Registers at call:**
AX = 0911h
SI = 4647h ('FG')
DI = 4A4Dh ('JM')
DS:DX -> ASCIZ command string (max 100 bytes, 0Dh OK)

**Return Registers:**
none

### INT 03h Function 0912h
*Soft-ICE*
### *BACK DOOR COMMANDS - GET BREAKPOINT INFORMATION*

**Registers at call:**
AX = 0912h
SI = 4647h ('FG')
DI = 4A4Dh ('JM')

**Return Registers:**
BH = entry number of last breakpoint set
BL = type of last breakpoint set
00h BPM (breakpoint register types)
01h I/O
02h INTerrupt
03h BPX (INT 03h-style breakpoint)
04h reserved
05h range
DH = entry number of last breakpoint to be triggered
DL = type of last triggered breakpoint (see above)

**See also:** Functions 0913h and 0914h

## INT 03h Function 0913h

*Soft-ICE v2.5x*

### *BACK DOOR COMMANDS - SET Soft-ICE BREAKPOINT*

**Registers at call:**
AX = 0913h
SI = 4647h ('FG')
DI = 4A4Dh ('JM')
DS:DX -> breakpoint structure (see Table 57-8)

**Return Registers:**
AX = status (00h,03h,06h,07h,09h,16h) (see Table 57-7)
00h successful
BX = breakpoint number

**See also:** Functions 0312h and 0314h

**57-7 Values for status:**

| | |
|---|---|
| 00h | successful |
| 03h | breakpoint table full |
| 06h | memory limit error |
| 07h | I/O limit error |
| 09h | range limit error |
| 16h | duplicate breakpoint |

**57-8 Format of breakpoint structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | breakpoint type (see Table 57-9) |
| 01h | DWORD | breakpoint address 1 (lower range limit for memory BPs, interrupt # for interrupt BPs, address of BP for execution BPs, I/O address (only word) for I/O BPs) |
| 05h | DWORD | breakpoint address 2 (upper range limit for memory BPs, optional value to check for for interrupt BPs, overlay number (0 = root) for execution BPs) |
| 09h | DWORD | breakpoint address 3 |
| 0Dh | BYTE | breakpoint mode 1 (see Table 57-10)<br>(for interrupt BPs, register to check:<br>00h no value checking, 01h check AL, 02h check AH, 03h check AX) |
| 0Eh | BYTE | breakpoint mode 2 (see Table 57-10) |
| 0Fh | BYTE | breakpoint size (00h byte, 01h word, 03h dword) |
| 10h | BYTE | breakpoint pass count before program stop |
| 11h | BYTE | breakpoint state |

**Note:** all unused fields should contain zeros

**57-9 Values for breakpoint type:**

| | |
|---|---|
| 00h | memory location |
| 01h | memory range |
| 03h | I/O |
| 04h | interrupt |
| 05h | execution break |

**57-10 Values for breakpoint mode:**

| | |
|---|---|
| 01h | read |
| 02h | write |
| 04h | execution |

## INT 03h Function 0914h

*Soft-ICE v2.5x*

### *BACK DOOR COMMANDS - REMOVE Soft-ICE BREAKPOINT*

**Registers at call:**
AX = 0914h
SI = 4647h ('FG')
DI = 4A4Dh ('JM')
BX = breakpoint number (returned by Function 0913h)

**Return Registers:**
BX = *unknown*

**See also:** Functions 0912h and 0913h

## Turbo Debugger

Turbo Debugger is the stand-alone debugger provided by Borland International with its high-end programming language products such as Borland C++ and Borland Pascal 7. TDX is Borland's Turbo Debugger variant for DPMI programs.

### INT 21h Function 3Fh

*Turbo Debug HARDWARE BREAKPOINTS*

### *READ STATUS BLOCK*

**Registers at call:**
AH = 3Fh
BX = handle for character device "TDHDEBUG"
CX = number of bytes to read
DS:DX -> buffer for status block (see Table 57-12)

**Return Registers:**
CF clear if successful
AX = number of bytes actually read
CF set on error
AX = error code (05h,06h) (see Table 24-2)

**See also:** Function 40h

.comment snake into two columns

**57-11 Values for status of command:**

| | |
|---|---|
| 00h | successful |
| 01h | invalid handle |
| 02h | no more breakpoints available |
| 03h | hardware does not support specified breakpoint type |
| 04h | previous command prevents execution |
| 05h | debugger hardware not found |
| 06h | hardware failure |
| 07h | invalid command |
| 08h | driver not initialized yet |
| FEh | recursive entry (hardware breakpoint inside hw bp handler) |

**57-12 Format of status block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | status of command (see Table 57-11) |
| **---status for command 01h---** | | |
| 01h | WORD | device driver interface version number (currently 1) |
| 03h | WORD | device driver software version |
| 05h | BYTE | maximum simultaneous hardware breakpoints |
| 06h | BYTE | configuration bits (see Table 57-13) |
| 07h | BYTE | supported breakpoint types (see Table 57-14) |
| 08h | WORD | supported addressing match modes (see Table 57-15) |
| 0Ah | WORD | supported data matches (see Table 57-16) |
| 0Ch | BYTE | maximum data match length (01h, 02h, or 04h) |
| 0Dh | WORD | size of onboard memory (in KB) |
| 0Fh | WORD | maximum number of trace-back events |
| 11h | WORD | hardware breakpoint enable byte address segment (0000h if not supported) |
| **---status for command 04h---** | | |
| 01h | BYTE | handle to use when referring to the just-set breakpoint |

**57-13 Bitfields for configuration bits:**

| | |
|---|---|
| bit 0 | CPU and DMA accesses are distinct |
| bit 1 | can detect DMA transfers |
| bit 2 | supports data mask |
| bit 3 | hardware pass counter on breakpoints |
| bit 4 | can match on data as well as addresses |

### 57-14 Bitfields for supported breakpoint types:

| | |
|---|---|
| bit 0 | memory read |
| bit 1 | memory write |
| bit 2 | memory read/write |
| bit 3 | I/O read |
| bit 4 | I/O write |
| bit 5 | I/O read/write |
| bit 6 | instruction fetch |

### 57-15 Bitfields for supported addressing match modes:

| | |
|---|---|
| bit 0 | any address |
| bit 1 | equal to test value |
| bit 2 | not equal |
| bit 3 | above test value |
| bit 4 | below test value |
| bit 5 | below or equal |
| bit 6 | above or equal |
| bit 7 | within range |
| bit 8 | outside range |

### 57-16 Bitfields for supported data matches:

| | |
|---|---|
| bit 0 | any data |
| bit 1 | equal to test value |
| bit 2 | not equal |
| bit 3 | above test value |
| bit 4 | below test value |
| bit 5 | below or equal |
| bit 6 | above or equal |
| bit 7 | within range |
| bit 8 | outside range |

---

## INT 21h Function 40h

*Turbo Debug HARDWARE BREAKPOINTS*

### *SEND COMMAND TO HARDWARE BREAKPOINT DRIVER*

**Registers at call:**

AH = 40h
BX = handle for character device "TDHDEBUG"
CX = number of bytes to write
DS:DX -> hardware breakpoint command
(see Table 57-17)

**Return Registers:**

CF clear if successful
AX = number of bytes actually written
CF set on error
AX = error code (05h,06h) (see Table 24-2)

**Details:** Results are retrieved by reading from the device.

**See also:** Function 3Fh

### 57-17 Format of hardware breakpoint commands:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | command code<br>00h install interrupt vectors<br>01h get hardware capabilities<br>02h enable hardware breakpoints<br>03h disable hardware breakpoints<br>04h set hardware breakpoint<br>05h clear hardware breakpoint<br>06h set I/O base address and reset hardware<br>07h restore interrupt vectors |

---command code 00h---

| | | |
|---|---|---|
| 01h | DWORD | pointer to Turbo Debugger entry point to be jumped to on hardware breakpoint; call with CPU state the same as on the breakpoint except for pushing AX and placing an entry code (FFh if breakout button or breakpoint handle) in AH |

---command code 04h---

| | | |
|---|---|---|
| 01h | BYTE | breakpoint type<br>00h memory read, 01h memory write, 02h memory read/write,<br>03h I/O read, 04h I/O write, 05h I/O read/write,<br>06h instruction fetch |
| 02h | BYTE | address matching mode<br>00h match any,<br>01h equal to test value, 02h different from test value,<br>03h above test value, 04h below test value,<br>05h below or equal to test value,<br>06h above or equal to test value,<br>07h within inclusive range, 08h outside specified range |
| 03h | DWORD | 32-bit linear low address |
| 07h | DWORD | 32-bit linear high address |
| 0Bh | WORD | pass count |
| 0Dh | BYTE | data size (01h, 02h, or 04h) |
| 0Eh | BYTE | source of matched bus cycle (01h CPU, 02h DMA, 03h either) |
| 0Fh | BYTE | data-matching mode (see address matching mode above) |
| 10h | DWORD | low data value |
| 14h | DWORD | high data value |
| 18h | DWORD | data mask specifying which bits of the data are tested |

---command code 05h---

| | | |
|---|---|---|
| 01h | BYTE | handle of breakpoint to clear (breakpoint returned from command 04h) |

---command code 06h---

| | | |
|---|---|---|
| 01h | WORD | base address of hardware debugger board |

### INT 2Fh Function FB43h, Subfunction 0100h
### *INSTALLATION CHECK*

*Borland TDX*
*Protected Mode (undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FB43h<br>BX = 0100h | BX = FB43h if loaded |

**Details:** Borland Pascal 7 DPMI programs use this call to check whether they should install their own stack and general protection exception handlers, or allow TDX to handle those exceptions.

### INT FEh
### *OVERLAY MANAGER*

*Turbo Debugger 8086 v2.5+*

**See also:** INT 3Fh

## Miscellaneous

### INT 00h
### *ROM DEBUGGER*

*Zenith*

**Purpose:** This interrupt invokes the ROM Debugger when at the BIOS level; it is equivalent to pressing Ctrl-Alt-Ins on booting.
**Details:** Since DOS revectors INT 00h, it is necessary to restore this vector to its original ROM BIOS value in order to invoke the debugger once DOS loads.
**See also:** Columbia INT 03h

### INT 03h

*Columbia PCs (desktop, VP portables)*

***ROM DEBUGGER***

**Purpose:** This interrupt invokes the ROM Debugger if INT 03h has not been revectored; it is equivalent to pressing Esc on booting.
**See also:** Zenith INT 00h

### INT 21h Function 3022h

*StopPrg v2.0*

***INSTALLATION CHECK***

**Program:** StopPrg is a resident program aborter by MAK-TRAXON's Prophet.

**Registers at call:**
AX = 3022h

**Return Registers:**
AX = DOS version (see MS-DOS Function 30h, Chapter 24)
CX = 1112h if StopPrg installed
BX = segment of resident code

**Details:** StopPrg may be temporarily disabled by storing 9090h in the word at 0000h:04FEh.
**Conflicts:** MS-DOS (Chapter 24)

### INT 2Fh Function 86h

*Unknown*
*(undocumented)*

*UNDETERMINED FUNCTION*

**Registers at call:**
AH = 86h
AL = function (at least 06h and 07h)
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** This function is called by Codeview for Windows.
**See also:** Function 44h

### INT 2Fh Function D600h

*HEART.COM*

***INSTALLATION CHECK***

**Program:** HEART.COM is a CPU lock-up/critical indicator utility by Mitch Davis.

**Registers at call:**
AX = D600h

**Return Registers:**
AX = 0303h (two hearts) if installed
ES:DI -> buffer (see Table 57-18)

**Details:** Once the host program has identified the address of the data area, it can change this to indicate safe/critical, alternate colours, etc.

The entries for the color table are in char/attrib form. Every two entries form a pair which is alternated between 68 times a minute. The first half of the table is for color videos, the second mono. Within each half, the first half is for the safe chars, and the second for the critical chars.
**Conflicts:** Vedit VSWAP (Chapter 66); see also Table 2-2

**57-18 Format of buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 8 WORDs | table of colors/attributes (see notes above) |
| 10h | BYTE | flags<br>bit 0: program is in critical section, so flash double exclamation mark<br>bit 1: program is in safe code, so flash the heart character |
| 11h | WORD | position of heartbeat on screen, normally 009Eh (last column of second line) |

### INT 2Fh Function DA55h

*TRAP.COM*
*(undocumented)*

***INSTALLATION CHECK***

**Program:** TRAP is an interrupt call tracer by Patrick Phillipot/Udo Chrosziel.

**Registers at call:**
AX = DA55h
DL = interrupt number
DH = *unknown*

**Return Registers:**
if installed
AH = interrupt number
AL = *unknown*
ES:BX -> *unknown*

**Details:** A separate copy of TRAP is loaded for each interrupt to be traced; thus the interrupt number is part of the installation check.
**Conflicts:** none, but see Table 2-2

---

### INT 41h Function 12h
### *"OutputDebugString"*

*MS Windows debugging kernel*
*Protected Mode*

| Registers at call: | Return Registers: |
|---|---|
| AH = 12h | *unknown* |
| *others, if any, unknown* | |

**Conflicts:** Acorn BBC Master (Chapter 7), Z100 (Chapter 7), Hard Disk (Chapter 10)
**See also:** Function 50h

---

### INT 41h Function 50h
### *"DefineDebugSegment"*

*MS Windows debugging kernel*
*Protected Mode*

| Registers at call: | Return Registers: |
|---|---|
| AH = 50h | *unknown* |
| *others, if any, unknown* | |

**Conflicts:** Acorn BBC Master (Chapter 7), Z100 (Chapter 7), Hard Disk (Chapter 10)
**See also:** Function 12h

---

### INT 60h

*INTRSPY/CMDSPY API*
*(undocumented)*

**Program:** INTRSPY is a script-driven debugger included with the book *Undocumented DOS.*
**Details:** INTRSPY will hook the first available interrupt in the range 60h-67h.

The installation check is to a) determine that the handler is an IRET instruction b) the signature 0Dh "INTRSPY vN.NN" immediately precedes the handler If INTRSPY is installed, the DWORD immediately after the IRET stores its entry point.
**Conflicts:** see Table 2-3

### 57-19 INTRSPY entry point calling convention:

**Registers at call:**
AH = function
- 00h *unknown*
- 01h set current directory (for use in reporting)
  - ES:DI -> counted string containing directory name (max 79 char)
- 02h set name of script file
  - ES:DI -> counted string containing file name (max 79 chars)
- 03h set script arguments
  - ES:DI -> counted string containing arguments (max 79 chars)
- 04h get directory set with function 01h
  - ES:DI -> 80-byte buffer for directory name
- 05h get name of script file
  - ES:DI -> 80-byte buffer for script filename
- 06h get script arguments
  - ES:DI -> 80-byte buffer for script arguments
- 07h get *unknown value*
  - CL = 00h-15h specifies what to get
  - ES:DI -> WORD to be set with desired value on return
- 08h get *unknown value*
  - ES:DI -> WORD to be set with returned value
- 09h get *unknown value*
  - ES:DI -> WORD to be set with returned value
- 0Bh store *code for interrupt handler*

  - ES:DI -> data
  - CX = number of bytes
- 0Ch *unknown*
  - ES:DI -> *unknown item*
- 0Dh get *unknown value*
  - ES:DI -> BYTE to be set with returned value
- 0Eh set *uknown* flag
- 0Fh clear *unknown* flag
- 10h *unknown*
- 11h *unknown*
- 12h get *unknown*
  - ES:DI -> buffer
- 13h *unknown*

**Return Registers:**

AH = 00h

AL = status

- 00h successful
- 01h invalid function
- 02h *unknown*
- 03h *unknown*
- 04h *unknown*
- 05h *unknown*

**---function 12h---**

CX = number of bytes returned in buffer

---

### INT 69h Function 6996h

*ISR.COM v1.00*

### *SPECIFY INTERRUPT HANDLER*

**Program:** ISR (Interrupt Service Reflector) is a TSR by Rich Bono which permits a program to provide hardware interrupt handlers even while being debugged with a debugger that swaps interrupt vectors during debugging.

**Registers at call:**

AX = 6996h

DS:DX -> interrupt handler or 0000h:0000h to disable

**Return Registers:**

AX = 9669h

**Details:** The interrupt vector which is to be reflected is set at installation time and cannot be changed.

**Conflicts:** Zenith AT (Chapter 7), DECnet DOS (NI), 10Net (NI)

## CHAPTER 58

# *Remote Control Software*

One class of software that has become invaluable to consultants and others who must troubleshoot systems from a distance is that which permits one computer to take full control of another at a remote location. By using such systems, the troubleshooter can make a "house call" by telephone without having to travel anywhere.

## Carbon Copy Plus

Carbon Copy Plus is produced by Meridian Technology, Inc. All three of the Carbon Copy Plus functions described in this section conflict with the Oak VGA BIOS (Chapter 8), GO32 (Chapter 37), and TopView (Chapter 44).

---

### INT 10h Function FF00h

*CARBON COPY PLUS v5.0*

***CHECK IF CC CONNECTED TO CCHELP***

**Registers at call:**
AX = FF00h

**Return Registers:**
BL = status (00h not connected, 01h connected)

**See also:** Functions FF01h and FF02h

---

### INT 10h Function FF01h

*CARBON COPY PLUS v5.0*

***DISCONNECT AND RESET LINE***

**Registers at call:**
AX = FF01h

**Return Registers:**
none

**See also:** Functions FF00h and FF02h

---

### INT 10h Function FF02h

*CARBON COPY PLUS v5.0*

***GET LAST PHONE NUMBER DIALED***

**Registers at call:**
AX = FF02h

**Return Registers:**
ES:DI -> ASCIZ phone number

**See also:** Functions FF00h and FF01h

## COMMUTE

COMMUTE is a remote-control package by Gus Grubba which was licensed by Central Point Software and has been included with PC Tools Deluxe since version 7.0. As with all programs using interrupts in the range 60h to 67h, there are numerous potential conflicts, which are listed in Table 2-3.

---

### INT 62h Functions 47h to 4Ah

*PC Tools v7 COMMUTE*

***UNDETERMINED FUNCTIONS***

**Registers at call:**
AH = function

47h *unknown*
    AL = subfunction (00h-31h)
48h *unknown*
    AL = *unknown*
49h *unknown*
4Ah *unknown*
    AL = subfunction (00h-46h)
CF set
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Function 49h may be the same as Function 4Ch.

---

### INT 62h Function 4Bh
### *UNDETERMINED FUNCTION*

*PC Tools v7 COMMUTE*

**Registers at call:**
AH = 4Bh
BX = 1234h
CX = 1234h
ES = *unknown*
CF set

**Return Registers:**
*unknown*

---

### INT 62h Function 4Ch
### *UNDETERMINED FUNCTIONS*

*PC Tools v7 COMMUTE*

**Registers at call:**
AH = 4Ch
BL = subfunction
    00h *unknown*
    02h *unknown*

**Return Registers:**
CF clear if successful
CF set on error

---

### INT 62h Function 6262h
### *INSTALLATION CHECK*

*PC Tools v7 COMMUTE*

**Registers at call:**
AX = 6262h
CF set

**Return Registers:**
AX = 0000h
BX = segment of resident code's PSP

## pcANYWHERE

pcANYWHERE is a remote control program from Dynamic Microprocessor Associates which was recently acquired by Symantec, the makers of the Norton Utilities (Chapter 62) and other products. This heritage can still be seen in the installation check, INT 21h Function 2B44h.

---

### INT 16h Function 75h
### *SET TICK COUNT FOR SCANNING*

*pcANYWHERE III*

**Registers at call:**
AH = 75h
AL = number of ticks between checks for new screen changes

**Return Registers:**
none

---

### INT 16h Function 76h
### *SET ERROR CHECKING TYPE*

*pcANYWHERE III*

**Registers at call:**
AH = 76h
AL = error checking type (00h none, 01h fast, 02h slow)

**Return Registers:**
none

**INT 62h Function 4Bh**

### INT 16h Function 77h
*pcANYWHERE III*

***LOG OFF***

**Registers at call:**
AH = 77h
AL = mode
- 00h wait for another call
- 01h leave in Memory Resident Mode
- 02h leave in Automatic Mode
- FFh leave in current operating mode

**Return Registers:**
none

**Conflicts:** WATCH (Chapter 66), PUSHDIR (Chapter 66)

### INT 16h Function 79h
*pcANYWHERE III*

***CHECK STATUS***

**Registers at call:**
AH = 79h

**Return Registers:**
AX = status
- FFFFh if resident and active
- FFFEh if resident but not active
- FFFDh if in Memory Resident mode
- FFFCh if in Automatic mode
- other value if not resident

**See also:** Function 7B00h, INT 21h Function 2B44h

### INT 16h Function 7Ah
*pcANYWHERE III*

***CANCEL SESSION***

**Registers at call:**
AH = 7Ah

**Return Registers:**
none

### INT 16h Function 7B00h
*pcANYWHERE III*

***SUSPEND***

**Registers at call:**
AX = 7B00h

**Return Registers:**
none

**See also:** Functions 79h and 7B01h

### INT 16h Function 7B01h
*pcANYWHERE III*

***RESUME***

**Registers at call:**
AX = 7B01h

**Return Registers:**
none

**See also:** Functions 79h and 7B00h

### INT 16h Function 7Ch
*pcANYWHERE III*

***GET PORT CONFIGURATION***

**Registers at call:**
AH = 7Ch

**Return Registers:**
AH = port number
AL = baud rate (see Table 58-1)

**58-1 Baud rates:**

| | | |
|---|---|---|
| 00h | 50 | baud |
| 01h | 75 | baud |
| 02h | 110 | baud |
| 03h | 134.5 | baud |
| 04h | 150 | baud |
| 05h | 300 | baud |
| 06h | 600 | baud |
| 07h | 1200 | baud |

| | | |
|---|---|---|
| 08h | 1800 | baud |
| 09h | 2000 | baud |
| 0Ah | 2400 | baud |
| 0Bh | 4800 | baud |
| 0Ch | 7200 | baud |
| 0Dh | 9600 | baud |
| 0Eh | 19200 | baud |

---

### INT 16h Function 7Dh

### *GET/SET TERMINAL PARAMETERS*

*pcANYWHERE III*

**Registers at call:**
AH = 7Dh
AL = subfunction
    00h set terminal parameters
    01h get terminal parameters
    02h get configuration header and terminal parameters
DS:CX -> terminal parameter block

**Return Registers:**
none

---

### INT 16h Function 7Eh

### *COMMUNICATIONS I/O THROUGH PORT*

*pcANYWHERE III*

**Registers at call:**
AH = 7Eh
AL = subfunction
    01h port input status
        Return AX = 0 if no characer ready,
        AX = 1 if character ready
    02h port input character
        Return AL = received character
    03h port output character in CX
    11h hang up phone

---

### INT 16h Function 7Fh

### *SET KEYBOARD/SCREEN MODE*

*pcANYWHERE III*

**Registers at call:**
AH = 7Fh
AL = subfunction
    00h enable remote keyboard only
    01h enable host keyboard only
    02h enable both keyboards
    08h display top 24 lines
    09h display bottom 24 lines
    10h Hayes modem
    11h other modem
    12h direct connect

**Return Registers:**
none

---

### INT 21h Function 2B44h

### *INSTALLATION CHECK*

*pcANYWHERE IV/LAN*

**Registers at call:**
AX = 2B44h ('D')
BX = 4D41h ('MA')
CX = 7063h ('pc')
DX = 4157h ('AW')
**See also:** INT 16h Function 79h

**Return Registers:**
AX = status
    4F4Bh ('OK') if large host resident
    6F6Bh ('ok') if small host resident
CX:DX -> API entry point

### 58-2 API entry point calling convention:

| Registers at call: | Return Registers: |
|---|---|
| AX = 0000h get pcANYWHERE IV version<br>DS:SI -> BYTE buffer for host type code | AH = version number<br>AL = revision number<br>DS:DI buffer byte filled with<br>00h full-featured host<br>01h limited-feature LAN host<br>other API may not be supported |
| AX = 0001h initialize operation<br>DS:SI -> initialization request structure (see Table 58-3) | AX = function status (see Table 58-5) |
| AX = 0002h get status | AH = current operating mode (see Table 58-3)<br>AL = current connection status<br>bit 0: a physical connection is active<br>bit 1: remove screen updating is active<br>bit 2: connection checking is active<br>bit 3: hot key detection is active<br>bit 4: background file transfer is active |
| AX = 0003h suspend remote screen updates | AX = function status (see Table 58-5) |
| AX = 0004h resume screen updates | AX = function status (see Table 58-5) |
| AX = 0005h end current remote access session<br>DS:SI -> termination request structure (see Table 58-4) | AX = function status (see Table 58-5) |
| AX = 0006h remove pcANYWHERE IV from memory | AX = status (0000h,FFD1h,FFD2h) (see Table 58-5) |
| AX = 8000h read data from communications channel<br>DS:BX -> buffer<br>CX = buffer size | AX >= number of characters read/available<br>AX < 0 on error (see Table 58-5) |
| AX = 8001h write data to communications channel<br>DS:BX -> buffer<br>CX = buffer size | AX >= number of characters written<br>AX < 0 on error (see Table 58-5) |
| AX = 8002h get connection status | AX = status<br>0000h if connection lost<br>> 0000h if connection active<br>< 0000h on error (see Table 58-5) |

### 58-3 Format of initialization request structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | operating mode<br>00h wait for a call<br>01h hot key activates<br>02h incoming call activates<br>03h initiate a call |
| 01h | 3 BYTEs | user ID to append to config file names |
| 04h | WORD | DS-relative pointer to path for config files |
| 06h | WORD | DS-relative pointer to path for program files |

### 58-4 Format of termination request structure:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | operating mode after termination<br>00h wait for a call, 01h hot key activates<br>02h incoming call activates, 80h use current mode<br>FFh remove from memory |

### 58-5 Values for function status:

| | |
|---|---|
| 0000h | function completed successfully |
| FFD1h | unable to release interrupt vectors |
| FFD2h | unable to release allocated memory |
| FFF2h | unable to establish a connection when operating mode is "Initiate a call" |
| FFF3h | modem configuration is invalid (corrupt configuration) |
| FFF4h | modem initialization failed (no modem response) |
| FFF5h | the communications device could not be initialized |
| FFF6h | the host operator aborted the function |
| FFF7h | the communications driver type specified in the configuration file is different than the one loaded when pcANYWHERE IV was initially started |
| FFF9h | the configuration file is invalid |
| FFFAh | the configuration file could not be found |
| FFFBh | no session is active |
| FFFCh | a remote access session is active |
| FFFDh | the specified operating mode is invalid |

## Miscellaneous

### INT 2Dh Function 10h

*RATSR 2.0+*

***GET STATUS***

**Program:** RATSR is a utility by James Berry provided with RemoteAccess/Professional, a commercial bulletin board system, that allows remote control of a station over a network.

**Registers at call:**

AL = 10h

AH = AMIS multiplex number for RATSR (see Chapter 50)

**Return Registers:**

AL = status

- 01h listening (no connection)
- 02h receiving
- 03h sending station being monitored
- 04h init receive

AH = keyboard lock status (00h unlocked, 01h locked)

**See also:** AMIS INT 2Dh (Chapter 50)

## INT 2Fh Function D300h

### *INSTALLATION CHECK*

*TeleReplica*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = D300h | SI = segment of resident code |
| BX = 4562h | AX = 251Dh |
| CX = 2745h | BX = DF21h |
| DX = *serial port I/O base address (03F8h for v3.9)* | CX = F321h |
| | DX = *unknown* |

**Conflicts:** none, but see Table 2-2

**See also:** LapLink Function D3CBh (Chapter 12)

## CHAPTER 59

# Viruses and Anti-Viral Tools

One of the less attractive features accompanying the free sharing of data files and programs within the microcomputer community has been the appearance of "virus" programs. From a class of programs so rare that many active participants in on-line data exchange considered them to be mythical not many years ago, the virus phenomenon has now become so commonplace that it has given birth to an industry devoted to anti-viral programs. In this chapter, we describe both the disease and some of the cures.

## Viruses

We have mixed feelings about providing publicity to the virus programs themselves; however, in the interest of providing as complete coverage as possible of the PC's interrupt usage and to facilitate creation of new anti-viral software, we have included all those about which we have information.

### 59-1 Virus installation checks on INT 21h:

| *Registers on call* | *Return Values* | *Virus* |
|---|---|---|
| AX = 0B56h | AX = 4952h | Perfume |
| AX = 0D20h | AX = 1971h | Crazy Imp |
| AH = 30h, DX = ABCDh | DX = DCBAh | Possessed |
| AX = 30F1h | AL = 00h | Dutch-555/Quit-1992 |
| AX = 330Fh | AL = 0Fh (DOS returns AL=FFh) | Burghofer |
| AX = 33E0h | AL = E0h (DOS returns AL=FFh) | Oropax |
| AX = 357Fh | DX = FFFFh | Agiplan/Month 4-6 |
| AX = 3DFFh | AX = 4A44h | JD-448 |
| AX = 4203h | AX = 1234h | Shake |
| AX = 4243h | AX = 5678h | Invader |
| AX = 4B04h | CF clear | MG |
| AX = 4B04h | AX = 044Bh | 669/Thirteen Minutes |
| AX = 4B25h | DI = 1234h | 1063/Mono |
| AX = 4B40h | AX = 5678h | Plastique/AntiCad |
| AX = 4B4Ah | AL = 57h | Jabberwocky |
| AX = 4B4Bh | CF clear | Horse-2 |
| AX = 4B4Dh | CF clear | Murphy-2 |
| AX = 4B4Dh | CF clear | Patricia/Smack |
| AX = 4B50h | AX = 1234h | Plastique-2576/AntiCad-2576 |
| AX = 4B53h | CF clear | Horse |

| | | |
|---|---|---|
| AX = 4B55h | AX = 1231h | Sparse |
| AX = 4B59h | CF clear | Murphy-1 |
| AX = 4B59h | CF clear | Murphy-4 |
| AX = 4B5Eh | CF clear | Brothers |
| AX = 4B87h | AX = 6663h | Shirley |
| AX = 4B95h | AX = 1973h | Zherkov-1882 |
| AX = 4BA7h | AX = B459h | 1876/Dash-em |
| AX = 4BAAh | CF clear | Nomenklatura |
| AX = 4BAFh | AL = AFh | Magnitogorsk |
| AX = 4BAFh | AL = FAh | 948/Screenplus1 |
| AX = 4BDDh | AX = 1234h | Lozinsky/Zherkov |
| AX = 4BFEh | AX = 1234h | Hitchcock |
| AX = 4BFEh | AX = ABCDh | 1193/Copyright |
| AX = 4BFEh | DI = 55BBh | Dark Avenger-1028 |
| AX = 4BFFh | BL = FFh | USSR-707 |
| AX = 4BFFh | DI = 55AAh | Justice |
| AX = 4BFFh | CF clear | Europe 92 |
| AX = 4BFFh, SI = DI = 0000h | DI = 55AAh | Cascade |
| AX = 5252h | BX = FFEEh | 516/Leapfrog |
| AX = 58CCh | CF clear | 1067/Headcrash |
| AX = 6969h | AX = 0666h | Rape-747 |
| AH = 76h | AL = 48h | Klaeren/Hate |
| AX = 7700h | AX = 0920h | Growing Block |
| AH = 7Fh | AH = 80h | Squeaker |
| AH = 83h | DX = 1990h | SVC |
| AH = 84h | DX = 1990h, BH=version | SVC 5.0 or SVC 6.0 |
| AH = 89h | AX = 0123h | Vriest |
| AH = 90h | AH = 01h | Carioca |
| AX = 9753h | never, execs original program | Nina |
| AX = A1D5h | AX = 900Dh | 789/Filehider |
| AX = A55Ah | AX = 5AA5h | Eddie-2 |
| AX = AA00h | AX = 00AAh | Blinker |
| AX = AA03h | AX = 03AAh | Backtime |
| AH = ABh | AX = 5555h | 600 or Voronezh-family |
| AX = BBBBh | AX = 6969h | Hey You |
| AH = BEh | AX = 1234h | Datalock |
| AX = BE00h, CF set | CF clear | USSR-1049 |
| AH = C0h | AX = 0300h | Slow/Zerotime |
| AH = C0h | AX = 1234h | Solano |
| AX = C301h, DX = F1F1h | DX = 0E0Eh | 905/Backfont |
| AX = C500h | AX = 6731h | Sverdlov |
| AH = C6h | AL = 55h | Socha |
| AX = C603h, CF set | CF clear | Yankee or MLTI |
| AX = C700h | AL = 07h | MH-757 |
| AH = CBh | AL = 07h | Milous |
| AX = CB02h | AX = 02CBh | Witcode |
| AH = CCh | AX = 0700h | Westwood |

| | | |
|---|---|---|
| AX = D000h | BX = 1234h | Fellowship |
| AX = D5AAh | AX = 2A55h | Diamond-A |
| AX = D5AAh | AX = 2A03h | Diamond-B family |
| AX = D5AAh, BP = DEAAh | SI = 4321h | Dir |
| AX = DADAh | AH = A5h | Gotcha |
| AX = DAFEh | AX = 1234h | Plovdiv 1.3 |
| AH = DEh | AH = DFh | Durban |
| AX = DEDEh | AH = 41h | Brothers |
| AH = E0h | AX = 0300h | Jerusalem |
| AH = E0h | AX = DADAh | Armagedon |
| AX = E00Fh | AX = 4C31h | 8-tunes |
| AH = E1h | AX = 0300h | Mendoza |
| AH = E1h | AX = 0400h | Fu Manchu |
| AH = E4h | AH = 04h | Anarkia |
| AH = E7h | AH = 78h | Spyer/Kiev |
| AX = EC59h | BP = EC59h | Terror |
| AH = EEh | AX = 0300h | Jerusalem-G |
| AH = EEh | AL = 05h | Pregnant |
| AH = F0h | AX = 0300h | Frere Jacques |
| AX = F1E9h | AX = installation state (see separate entry below) | Tremor |
| AX = F2AAh | AH = AAh | PcVrsDs |
| AH = F3h | AX = 0400h | Carfield |
| AH = F7h | AX = 0300h | GP1 |
| AH = FBh | AH = 00h | Cinderella |
| AX = FB0Ah | AX = 0AFBh | dBASE |
| AH = FCh | AL = A5h | Troi |
| AH = FDh | AH = 13h | Border |
| AH = FEh | AH = 00h | 483 |
| AX = FE01h | AX = 01FEh | Flip |
| INT 21h;Function FFFFh | | |
| AX = FE02h | AX = 01FDh | 2468/Tequila |
| AX = FEDCh | AL = DCh | Black Monday |
| AH = FFh | AH = 00h | Tumen 0.5 or Hero |
| AH = FFh | AX = 0400h | Sunday |
| AX = FF10h | AL = 07h | Twins |
| AX = FFFEh | AX = 0815h | 08/15 (Many Fingers) |
| AX = FFFFh | AX = 0000h | Ontario |
| AX = FFFFh | AX = 1992h | Year 1992/B1M92 |
| AX = FFFFh, CX = 0000h | CX = 0006h | Revenge |

**Note:** While few of these virus installation checks conflict directly with MS-DOS, many conflict with third-party software such as Novell NetWare and DoubleDOS.

**59-2 Unknown viral INT 21h functions:**

| *Call* | *Used by* | *Comments* |
|---|---|---|
| AX = 4B20h | Holocaust/Telefonica | |
| AX = 4B21h | Holocaust/Telefonica | called at completion of virus installation |
| AX = 4B41h | Plastique/AntiCad | |
| AX = 4B60h | Plastique-2576/AntiCad-2576 | |
| AH = C1h | Solano | |
| AH = C2h | Scott's Valley | |
| AX = CA15h | Piter | |
| AH = CDh | Westwood | |
| AH = D5h | Carfield | |
| AH = DEh | April 1st EXE | |
| AH = F1h | F1-337 | |

**Note:** It is not known whether any additional registers are used for input or what values are returned.

---

## INT 21h Function 58DDh

*VIRUS*

### *"1067"/"Headcrash"* - *GET ORIGINAL INT 21h VECTOR*

**Registers at call:**
AX = 58DDh

**Return Registers:**
CX = code segment of virus
ES:BX = old INT 21h vector

**See also:** Function 58CCh

---

## INT 21h Function DDh

*VIRUS*

### *"Jerusalem"-family* - *RELOCATE VIRUS*

**Registers at call:**
AH = DDh
CX = number of bytes to copy
DS:SI -> source of copy
ES:DI -> destination of copy

**Return Registers:**
does not return normally; return address is caller's CS:0100h with AX = *unknown*

**Conflicts:** Novell NetWare (NI)
**See also:** Functions E0h and EEh

---

## INT 21h Function F1E9h

*VIRUS*

### *"Tremor"* - *INSTALLATION CHECK*

**Registers at call:**
AX = F1E9h

**Return Registers:**
AX = installation state
CADEh installed, and calling program is infected
F100h not installed (normal DOS return value)
else installed, but calling program is not infected

**Conflicts:** DoubleDOS (Chapter 49), Novell NetWare (NI)
**See also:** Functions F1h and F2AAh

**INT 21h Function FE03h** *VIRUS*

***"2468"/"Tequila" - DISPLAY VIRUS MESSAGE***

| Registers at call: | Return Registers: |
|---|---|
| AX = FE03h | none |

**Conflicts:** PCMANAGE/DCOMPRES (Chapter 16), DOS 2.11 (Chapter 24), DoubleDOS (Chapter 49)

**See also:** Function FE02h

---

**INT 31h** *VIRUS*

***"Vacsina" series - INSTALLATION CHECK (NOT A VECTOR!)***

**Details:** If one of the Vacsina viruses is resident, the low byte of this interrupt still contains the last byte of the INT 30h CP/M JMP instruction, but the remaining three bytes are 7Fh 39h followed by the Vacsina version number.

**See also:** INT 21h Function FFFFh, INT 32h

---

**INT 32h** *VIRUS*

***reportedly used by "Tiny" Viruses***

**Conflicts:** Plovdiv virus below

**See also:** INT 21h Function FFFFh, INT 31h, INT 44h

---

**INT 32h** *VIRUS*

***"Plovdiv 1.3"/"Damage 1.3" - ORIGINAL INT 21h VECTOR***

**Conflicts:** Tiny virus above

**See also:** INT 31h, INT 9Eh

---

**INT 44h** *VIRUS*

***"Lehigh" - ORIGINAL INT 21h VECTOR***

**Conflicts:** Acorn BBC Master (Chapter 7), Z100 (Chapter 7), PCjr (Chapter 8), Novell NetWare (NI), IBM 3270PC (NI)

**See also:** INT 32h, INT 60h, INT 70h, INT 9Eh

---

**INT 60h** *VIRUS*

***"Zero Bug" - INSTALLATION CHECK***

**Purpose:** The "Zero Bug" virus hooks this vector. It considers itself installed if offset 103h of the handler's segment contains the bytes "ZE".

**Conflicts:** see Table 2-3

**See also:** INT 32h, INT 44h, INT 61h

---

**INT 61h** *VIRUS*

***"SEMTEX"/"Screen Trasher" - INT 21h SUBSTITUTE***

**Details:** The virus copies the original INT 21h vector into INT 61h.

**Conflicts:** see Table 2-3

**See also:** INT 21h, INT 60h, INT 6Bh

---

**INT 6Bh** *VIRUS*

***"Saddam" - ORIGINAL INT 21h VECTOR***

**Conflicts:** Tandy Schoolmate Plus (Chapter 66), DECnet DOS (NI), Novell NASI/NACS (NI), TelAPI (NI), NCSI (NI)

**See also:** INT 21h Function FFFFh, INT 61h, INT 70h

---

**INT 70h** *VIRUS*

***"Stupid" - ORIGINAL INT 21h VECTOR***

**Details:** The virus sets this interrupt to be the same as INT 21h, and then performs only INT 70h calls; since INT 70h is also generated by the real-time clock on the PC/AT and higher, this could cause random actions on an infected system.

**Conflicts:** Real-Time Clock IRQ8 (Chapter 3)

**See also:** INT 6Bh, INT 9Eh, INT E0h

---

**INT 87h** *VIRUS*

***"ZeroHunt" - VIRAL CODE (NOT A VECTOR!)***

**Details:** The ZeroHunt virus copies its resident code down to 0000h:021Ch and following.

**Conflicts:** APL*PLUS/PC (Chapter 53), ROM BASIC (Chapter 54)

**See also:** INT 8Bh

---

**INT 8Bh** *VIRUS*

***"ZeroHunt" - INSTALLATION CHECK (NOT A VECTOR!)***

**Details:** If the ZeroHunt virus is resident, this vector will contain either EE83h:019Bh (ZH-411) or EE83h:019Fh (ZH-415).

**Conflicts:** APL*PLUS/PC (Chapter 53), ROM BASIC (Chapter 54)

**See also:** INT 70h, INT 87h, INT 9Ch

---

**INT 9Ch** *VIRUS*

***"INT13" - ORIGINAL INT 13h VECTOR***

**Conflicts:** ROM BASIC (Chapter 54)

**See also:** INT 8Bh, INT 9Dh, INT 9Eh, INT 9Fh

---

**INT 9Dh** *VIRUS*

***"INT13" - ROM INT 13h ENTRY POINT***

**Details:** This vector is used by the virus to store the result of a call to INT 2Fh Function 13h.

**Conflicts:** ROM BASIC (Chapter 54)

**See also:** INT 9Ch, INT 9Eh, INT 9Fh, MS-DOS INT 2Fh Function 13h (Chapter 24)

---

**INT 9Eh** *VIRUS*

***"INT13" - ORIGINAL INT 21h VECTOR***

**Conflicts:** ROM BASIC (Chapter 54)

**See also:** INT 70h, INT 9Ch, INT 9Dh, INT E0h

---

**INT 9Fh** *VIRUS*

***"INT13" - STORAGE FOR USER INT 13h VECTOR***

**Details:** While it is infecting a file, the INT13 virus grabs INT 13h and uses this interrupt to store the existing INT 13h vector for later restoration.

**Conflicts:** ROM BASIC (Chapter 54)

**See also:** INT 9Ch, INT 9Dh

---

**INT E0h** *"Micro-128" VIRUS*

*UNDETERMINED FUNCTION*

**Details:** Micro-128 also overwrites the upper half of the interrupt table.

**Conflicts:** DR Multiuser DOS (Chapter 48), APL*PLUS/PC (Chapter 53), ROM BASIC (Chapter 54)

**See also:** INT 9Eh, INT F1h

---

**INT F1h and INT FFh** *"Violetta" VIRUS*

*UNDETERMINED FUNCTIONS*

**Details:** These interrupts are used but not chained by the virus.

**Conflicts:** AT BIOS (Chapter 4), Z100 (Chapter 7), AQUEDUCT (Chapter 12), SPEECH.COM (Chapter 22), QEMM-386 (Chapter 39)

**See also:** INT E0h

## Anti-Viral Tools

The listing of tools in this section is necessarily incomplete because several of the leading producers of anti-viral programs insist on complete secrecy about their programs' methods of operation, and one has even stated in a private message that should their interface be publicized in the Interrupt List, they would immediately change it. Although this is not an entirely paranoid viewpoint considering that at least one virus checker has been clandestinely infected and then distributed and another is specifically attacked by several viruses, in our opinion, any API which requires secrecy to remain effective is inherently insecure. History shows that such "security through obscurity" has never withstood attack (just consider that all of the undocumented information in this book was in essence "protected" with this method).

---

**INT 13h Function 5342h** *ScanBoot*

***INSTALLATION CHECK***

**Program:** ScanBoot is a virus-detection TSR by PanSoft.

| Registers at call: | Return Registers: |
|---|---|
| AX = 5342h ("SB") | CF clear if ScanBoot installed |
| CX = 0001h | AX = 0000h |
| DX = 0000h | CX = serial number ("SW" if shareware release) |
| | DX = version |
| | BX,SI,ES destroyed |

---

**INT 13h Function FAh** *PC Tools v8+ VSAFE, VWATCH*

***API***

| Registers at call: | Return Registers: |
|---|---|
| AH = FAh | varies by function |
| DX = 5945h | |
| AL = function (00h-07h) | |

**Details:** This API is identical to the ones on INT 16h Function FAh and INT 21h Function FAh, so it is listed in its entirety under INT 16h Function FA00h and following.

**See also:** INT 16h Function FA00h

---

**INT 16h Function FA00h** *PC Tools v8+ VSAFE, VWATCH (undocumented)*

*INSTALLATION CHECK*

| Registers at call: | Return Registers: |
|---|---|
| AX = FA00h | CF clear |
| DX = 5945h | DI = 4559h |
| | BX = *unknown* (2F00h or FFFFh) (VSAFE only) |

**See also:** INT 13h Function FAh, INT 21h Function FAh, INT 2Fh Function 6282h

### INT 16h Function FA01h
***UNINSTALL***

*PC Tools v8+ VSAFE, VWATCH*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = FA01h | CF clear if successful |
| DX = 5945h | DI = 4559h |

**See also:** Function FA00h

---

### INT 16h Functions FA02h and FA03h
*UNDETERMINED FUNCTIONS*

*PC Tools v8+ VSAFE, VWATCH*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| DX = 5945h | CF clear |
| AX = function | DI = 4559h |
| FA02h *undetermined*<br>BL = *unknown* | CL = old value of *unknown* |
| FA03h *undetermined* | AX = 0002h |

---

### INT 16h Functions FA04h and FA05h
***GET/SET*** *UNKNOWN VALUE*

*PC Tools v8+ VSAFE*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| DX = 5945h | CF clear |
| AX = function | DI = 4559h |
| FA04h get | BL = *unknown* |
| FA05h set<br>BL = *unknown* | no other registers |

**Details:** These functions are NOPs under VWATCH.

---

### INT 16h Functions FA06h and FA07h
***GET/SET*** *UNKNOWN VALUE*

*PC Tools v8+ VSAFE, VWATCH*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| DX = 5945h | CF clear |
| AX = function | DI = 4559h |
| FA06h get | BL = *unknown* |
| FA07h set<br>BL = *unknown* | no other registers |

**See also:** Function FA00h

### INT 21h Function 4BEEh
*F-DRIVER.SYS v1.14+*

### *GRAB INT 21h*

**Registers at call:**
AX = 4BEEh

**Return Registers:**
AX = status
1234h if grab successful
2345h if failed (INT 21h grabbed previously)

**Details:** F-DRIVER.SYS is part of the F-PROT virus/trojan protection package by Fridrik Skulason.

When called the first time, this function moves the INT 21h monitoring code from its original location in the INT 21h chain to be the first thing called by INT 21h. This is the mechanism used by F-NET.

**See also:** INT 2Fh Function 4653h

---

### INT 21h Function FAh
*PC Tools v7+ VDEFEND, VSAFE, VWATCH (undocumented)*

### *API*

**Registers at call:**
AH = FAh
DX = 5945h
AL = function (00h-02h for VDEFEND, 00h-07h for VSAFE and VWATCH)

**Return Registers:**
varies by function

**Details:** This API is identical to the API on INT 13h Function FAh and INT 16h Function FAh, so it is listed in its entirety only under INT 16h Function FA00h and following.

**Conflicts:** DOS 2.11 (Chapter 24), DoubleDOS (Chapter 49)

**See also:** INT 13h Function FA00h, INT 16h Function FA00h

---

### INT 21h Function FEE7h
*IBMAVSH v1.02*

### *INSTALLATION CHECK*

**Registers at call:**
AX = FEE7h
BX = 8C00h
CX = A2B3h

**Return Registers:**
BX = 9E06h if installed

**Program:** IBMAVSH is a resident virus scanner included with IBM DOS 6.1.

**Conflicts:** Virus (above), DOS 2.11 (Chapter 24), DoubleDOS (Chapter 49)

---

### INT 21h Function FF0Fh
*FLU_SHOT+ v1.83*

### *INSTALLATION CHECK*

**Program:** FLU_SHOT+ is an antivirus/antitrojan program by Ross M. Greenberg and Software Concepts Design.

**Registers at call:**
AX = FF0Fh

**Return Registers:**
AX = 0101h if resident

**Details:** The "PSQR/1720" virus calls this function to determine whether FLU_SHOT+ is present.

**Conflicts:** Virus (above), CED (Chapter 13), DOSED (Chapter 13), DOS 2.11 (Chapter 24), DOS/4GW (Chapter 35), GO32 (Chapter 37), TopWare NOS (NI)

**See also:** Functions FFh and FF10h

---

### INT 2Fh Function 4653h, Subfunction 0002h

*F-PROT v1.x only*

***F-LOCK.EXE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4653h | |
| CX = 0002h | |
| BX = operation | |
| 0000h installation check | AX = FFFFh |
| 0001h uninstall | AX,BX,ES destroyed |
| 0002h disable (v1.08 and below only) | |
| 0003h enable (v1.08 and below only) | |

**Details:** F-LOCK is part of the F-PROT virus/trojan protection package by Fridrik Skulason.
**Conflicts:** Windows/286 DOS Extender (Chapter 37)
**See also:** Function 4653h Subfunction 0003h, Function CA00h, INT 21h Function 4BEEh

---

### INT 2Fh Function 4653h, Subfunction 0003h

*F-PROT v1.x only*

***F-XCHK.EXE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4653h | |
| CX = 0003h | |
| BX = operation | |
| 0000h installation check | AX = FFFFh |
| 0001h uninstall | AX,BX,ES destroyed |

**Details:** F-XCHK is part of the F-PROT virus/trojan protection package by Fridrik Skulason.
**Conflicts:** Windows/286 DOS Extender (Chapter 37)
**See also:** Function 4653h Subfunctions 0002h and 0004h, Function CA00h

---

### INT 2Fh Function 4653h, Subfunction 0004h

*F-PROT v1.x only*

***F-POPUP.EXE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4653h | |
| CX = 0004h | |
| BX = subfunction | |
| 0000h installation check | AX = FFFFh |
| 0001h uninstall | AX,BX,ES destroyed |
| 0002h disable (v1.08 and below only)<br>display message (v1.14+)<br>other registers: *unknown* | |
| 0003h enable (v1.08 and below only)<br>display message (v1.14+)<br>other registers: *unknown* | AX = key pressed by user (v1.14+) |

**Details:** F-POPUP is part of the F-PROT virus/trojan protection package by Fridrik Skulason.
**Conflicts:** Windows/286 DOS Extender (Chapter 37)
**See also:** Function 4653h Subfunctions 0003h and 0005h, Function CA00h

---

**INT 2Fh Function 4653h, Subfunction 0005h** *F-PROT v1.x only*

***F-DLOCK.EXE***

| Registers at call: | Return Registers: |
|---|---|
| AX = 4653h | |
| CX = 0005h | |
| BX = subfunction | |
| 0000h installation check | AX = FFFFh |
| 0001h uninstall | AX,BX,ES destroyed |

**Details:** F-DLOCK is part of the F-PROT virus/trojan protection package by Fridrik Skulason.
**Conflicts:** Windows/286 DOS Extender (Chapter 37)
**See also:** Function 4653h Subfunction 0004h, Function CA00h

---

**INT 2Fh Function 6282h** *PC Tools v7.0+ VDEFEND, VSAFE, VWATCH, DATAMON (undocumented)*

***SET** UNKNOWN **ADDRESS***

| Registers at call: | Return Registers: |
|---|---|
| AX = 6282h | BX = 0062h |
| CX:DX -> *unknown* or 0000h:0000h | |
| DI = address | |
| 0000h *unknown* | |
| FFFFh *unknown* | |
| other segment of *unknown* | |

**Details:** If CX:DX = 0000h:0000h on entry, the unknown address is not changed (DATAMON only).
**See also:** INT 13h Function FAh, INT 21h Function FAh

---

**INT 2Fh Function C900h** *ThunderByte (undocumented)*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = C900h | AL = FFh if installed |
| BP = 0000h | BP >= 0014h |

**Details:** This function is called by TBSCANX.
**Conflicts:** none, but see Table 2-2
**See also:** Functions C987h and CA00h

---

**INT 2Fh Function C987h** *ThunderByte (undocumented)*

*DISINFECT FILE*

| Registers at call: | Return Registers: |
|---|---|
| AX = C987h | AX = status |
| BX:DX -> filename | 0000h *successful* |
| BX:CX -> virus name | |

**Details:** This function is called by TBSCANX.
**Conflicts:** none, but see Table 2-2
**See also:** Function CA00h

## INT 2Fh Function CA00h

TBSCANX

### INSTALLATION CHECK

**Program:** TBSCANX is a resident virus scanning module by Frans Veldman.

**Registers at call:**
AX = CA00h
BX = 5442h ('TB')

**Return Registers:**
AL = status
00h not installed
FFh installed
BX = 7462h ('tb') if BX was 5442h on entry

**Details:** Programs may perform virus checks on themselves, other program files, or their data files by invoking the TBSCANX API.

**Conflicts:** see Table 2-2

**See also:** Functions 4653h and C900h

## INT 2Fh Function CA01h

TBSCANX

### GET STATUS

**Registers at call:**
AX = CA01h

**Return Registers:**
AH = BCD version number (v2.2+)
CAh for versions before 2.2
AL = state (00h = disabled, 01h = enabled)
CX = number of signatures which will be searched
**---v2.0---**
BX = EMS handle, 0000h if not using EMS
**---v2.3+---**
BX = segment of swap area, 0000h if not swapped
DX = EMS or XMS handle (XMS handle if BX=0000h), FFFFh if not using EMS

**Conflicts:** see Table 2-2

**See also:** Function CA02h

## INT 2Fh Function CA02h

TBSCANX

### SET STATE

**Registers at call:**
AX = CA02h
BL = new state (00h = disabled, 01h = enabled)

**Return Registers:**
none

**Conflicts:** see Table 2-2

**See also:** Function CA01h

## INT 2Fh Function CA03h

TBSCANX

### SCAN BUFFER

**Registers at call:**
AX = CA03h
CX = size of buffer
DS:DX -> buffer containing data sto scan

**Return Registers:**
CF clear if no virus signatures found
BX,ES destroyed
CF set if signature found
ES:BX -> ASCIZ virus name (v2.3+)
DS:DX -> ASCIZ virus name (v2.0)
AX,CX,DX destroyed (v2.3+)
all other registers except CS:IP and SS:SP destroyed (v2.0)

**Conflicts:** see Table 2-2

**See also:** Function CA04h

---

## INT 2Fh Function CA04h

*TBSCANX*

### *SCAN FILE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = CA04h | CF clear if no virus signatures found |
| DS:DX -> filename | BX,ES destroyed |
| | CF set if signature found |
| | ES:BX -> ASCIZ virus name |
| | AX,CX,DX destroyed |

**Details:** This function requires at least 4K free memory.
**Conflicts:** see Table 2-2
**See also:** Function CA03h

## CHAPTER 60

# *ZIPKEY*

ZIPKEY is a resident ZIPcode database by Eric Isaacson. If it is installed, the string "ZIPKEY" is present at offset 75h in the INT B3h handler's segment, and the byte at 7Bh contains the API version number (00h for v1.x, 01h for v2.0).

The entire ZIPKEY API conflicts with IBM ROM BASIC (Chapter 54).

---

### INT B3h Function 70h

*ZIPKEY*

### *GET VERSION*

**Registers at call:**
AH = 70h

**Return Registers:**
AH = major version
AL = minor version
CL = number of states and territories in current database
DH = year of current database - 1900
DL = month of current database's file date

**Return Registers:**
AX destroyed

---

### INT B3h Function 71h

*ZIPKEY*

### *CONVERT TWO-LETTER ABBREVIATION TO STATE CODE*

**Registers at call:**
AH = 71h
BX = abbreviation, in either case (first letter in BL)

**Return Registers:**
CF set on error
    AL = FFh
CF clear if successful
    AL = ZIPKEY state code

**See also:** Function 72h

---

### INT B3h Function 72h

*ZIPKEY*

### *CONVERT STATE CODE TO TWO-LETTER ABBREVIATION*

**Registers at call:**
AH = 72h
BL = ZIPKEY state code

**Return Registers:**
CF set on error
    AX destroyed
CF clear if successful
    AX = abbreviation, in upper case

**See also:** Functions 71h and 73h

---

### INT B3h Function 73h

*ZIPKEY*

### *CONVERT STATE CODE TO STATE NAME*

**Registers at call:**
AH = 73h

BL = ZIPKEY state code
ES:DI -> buffer for name

**Return Registers:**
CF set on error
    AX destroyed
CF clear if successful
    ES:DI points one byte beyond end of name

**See also:** Function 72h

---

### INT B3h Function 74h

*ZIPKEY*

### *CONVERT ZIPCODE TO ASCII DIGITS*

**Registers at call:**
AH = 74h
DX = zipcode region (0-999)
CH = last two digits of zipcode (0-99)
ES:DI -> buffer

**Return Registers:**
CF set on error
    AX destroyed
CF clear if successful
    ES:DI points one byte beyond end of digit string

---

### INT B3h Function 75h

*ZIPKEY*

### *LOOK UP STATE CODE FOR ZIPCODE*

**Registers at call:**
AH = 75h
DX = zipcode region (0-999)
CH = last two digits of zipcode (0-99)

**Return Registers:**
CF set on error (zipcode not found)
    AL = suggested state code, FFh if none
CF clear if successful
    AL = ZIPKEY state code
    BX = area code (v2.0+)

**See also:** Functions 76h and 79h

---

### INT B3h Function 76h

*ZIPKEY*

### *LOOK UP CITY AND STATE FOR ZIPCODE*

**Registers at call:**
AH = 76h
DX = zipcode region (0-999)
CH = last two digits of zipcode (0-99)
ES:DI -> buffer for name

**Return Registers:**
CF set on error
    AL = suggested state code, FFh if none
    ES:DI buffer filled with suggested city name
CF clear if successful
    AL = ZIPKEY state code
    BX = area code (v2.0+)
    ES:DI points one byte beyond end of name

**See also:** Functions 75h, 78h, and 7Eh

---

### INT B3h Function 77h

*ZIPKEY*

### *PLAY BACK EXIT KEY FOR ENTRY WITH GIVEN ZIPCODE*

**Registers at call:**
AH = 77h
DX = zipcode region (0-999)
CH = last two digits of zipcode (0-99)
BX = 16-bit BIOS keycode for a
    defined ZIPKEY alternate exit key

**Return Registers:**
CF set on error
    AX destroyed
CF clear if successful
    zipcode specification as defined by the BX
        keystroke is placed in keyboard buffer, as if
        the user had popped up ZIPKEY and exited
        by pressing the key specified by BX

---

### INT B3h Function 78h

*ZIPKEY*

### *LOOK UP ZIPCODES FOR A GIVEN STATE AND CITY*

**Registers at call:**
AH = 78h
BL = ZIPKEY state code

DS:SI -> city name, terminated with 0Dh if complete name, 00h if prefix

**Return Registers:**
BH = number of matching entries (set to 51 if more than 50)
DX = zipcode region of first match (0-999)
CL = last two digits of first zipcode in the range (0-99)
CH = last two digits of last zipcode in the range (0-99)
AX destroyed

**See also:** Functions 79h and 7Ah

---

### INT B3h Function 79h

*ZIPKEY*

### *LOOK UP ZIPCODES FOR A GIVEN CITY*

**Registers at call:**
AH = 79h
BL = ZIPKEY state code of first state to search
DS:SI -> city name, terminated with 0Dh if complete name, 00h if prefix

**Return Registers:**
AL = ZIPKEY state code of first matching state
BH = number of matching entries (set to 51 if more than 50)
DX = zipcode region of first match (0-999)
CL = last two digits of first zipcode in first range (0-99)
CH = last two digits of last zipcode in first range (0-99)

**Details:** To find all matching cities, repeat the search with BL set to one more than the returned AL.

**See also:** Functions 78h and 7Ah

---

### INT B3h Function 7Ah

*ZIPKEY*

### *FETCH AN ENTRY FROM A PREVIOUS LOOKUP*

**Registers at call:**
AH = 7Ah
BL = case number (0 to one less than value returned in BH by lookup)

**Return Registers:**
AL = ZIPKEY state code
DX = zipcode region (0-999)
CL = last two digits of first zipcode in the range (0-99)
CH = last two digits of last zipcode in the range (0-99)

**See also:** Functions 78h and 79h

---

### INT B3h Function 7Bh

*ZIPKEY*

### *GET VALUES NEEDED TO SAVE ZIPKEY CONTEXT*

**Registers at call:**
AH = 7Bh

**Return Registers:**
BL = maximum number of characters for a city name
BH = ZIPKEY state code for last city-name search FFh if none
CX:DX = internal code identifying last city search
AX destroyed

**See also:** Function 7Ch

---

### INT B3h Function 7Ch

*ZIPKEY*

### *RESTORE ZIPKEY CONTEXT*

**Registers at call:**
AH = 7Ch
BL = maximum number of characters for a city name
BH = ZIPKEY state code for last city-name search

FFh if none
CX:DX = internal code returned by Function 7Bh

**Return Registers:**
CF set on error
CF clear if successful
AX destroyed

**See also:** Function 7Bh

---

### INT B3h Function 7Dh
### *REQUEST POP UP*

*ZIPKEY*

**Registers at call:**
AH = 7Dh
BL = index number to simulate pressing a hotkey
FFh for immediate popup with no playback on return

**Return Registers:**
CF set on error
AL = error code
FDh already busy with another request
FEh illegal function
CF clear if successful
AX destroyed
window popped up and was closed by the user

**See also:** Function 70h

---

### INT B3h Function 7Eh
### *GET NAME OF PRIMARY CITY FOR A ZIPCODE REGION*

*ZIPKEY*

**Registers at call:**
AH = 7Eh
DX = zipcode region (0-999)
ES:DI -> buffer for name

**Return Registers:**
CF set on error
AL = FFh region does not exist
CF clear if successful
AL = ZIPKEY state code
ES:DI points one byte beyond end of name

**See also:** Function 76h

---

### INT B3h Function 7Fh
### *ENABLE/DISABLE HOTKEYS*

*ZIPKEY*

**Registers at call:**
AH = 7Fh
BL = function
00h turn off hotkeys
01h turn on hotkeys
02h return hotkey status
03h toggle hotkey status

**Return Registers:**
AL = hotkey status
00h off
01h on

---

### INT B3h Function 80h
### *DETERMINE STATE FOR AREA CODE*

*ZIPKEY v2.0+*

**Registers at call:**
AH = 80h
BX = telephone area code (decimal)

**Return Registers:**
CF clear if successful
AL = ZIPKEY state code
DX = first ZIP region for state (03E8h if Canada)
CX = number of ZIP regions in state
CF set on error
AL = FFh
DX = 03E9h

## CHAPTER 61

# *PC Tools*

While PC Tools, the all-purpose DOS utility package from Central Point Software, was not the first such utility to reach the market, it has for years enjoyed the reputation of packing more punch into a single package than any of its competitors. Version 8.0 includes a powerful disk editor and a customizable utility that optimizes memory managers such as QEMM-386, in addition to the file and disk management utilities of past editions. Most of the utilities, including undelete, work across NetWare LANs.

This chapter describes the interrupt interfaces used by all versions of PC Tools and by many of its accompanying utilities. Other utilities are described in different chapters which are more appropriate: PC-CACHE in Chapter 17, CPTASK in Chapter 49, COMMUTE in Chapter 58, and virus protection in Chapter 59. DRIVEMAP is a network redirector and is therefore covered in *Network Interrupts.*

---

### INT 16h Function 6969h, Subfunction 6968h

*v5.1+ BACKTALK*

#### *UNHOOK*

| Registers at call: | Return Registers: |
|---|---|
| AX = 6969h | resident code unhooked, but not removed from memory |
| BX = 6968h | |

---

### INT 16h Function 6969h, Subfunction 6969h

*v5.1+ BACKTALK*

#### *INSTALLATION CHECK*

| Registers at call: | Return Registers: |
|---|---|
| AX = 6969h | DX nonzero if installed |
| BX = 6969h | BX = CS of resident code |
| DX = 0000h | DX = PSP segment of resident code |
| | DS:SI -> ASCIZ identification string "CPoint Talk" |

---

### INT 16h Function FE55h

*v8+ DATAMON, DRIVEMAP, CPSCHED, CPTASK*
*(undocumented)*

#### *UNDETERMINED FUNCTION*

| Registers at call: | Return Registers: |
|---|---|
| AX = FE55h | DX:AX -> *unknown* |
| CX = *segment of resident CPTASK or 0000h* | |
| DX = *unknown* | |
| *others, if any, unknown* | |

---

### INT 16h Function FEA4h

*v7+ CPSCHED/DESKTOP*
*(undocumented)*

#### *RESET UNKNOWN ITEM*

| Registers at call: | Return Registers: |
|---|---|
| AX = FEA4h | none |

**Details:** This function is identical to Function FFA4h, and is implemented by the same code in DESKTOP.
**See also:** Function FFA4h

---

### INT 16h Function FEC6h
### *ENABLE/DISABLE CPSCHED API*

*v7+ CPSCHED*
*(undocumented)*

**Purpose:** Specify whether CPSCHED API calls other than this one and Function FE55h will be honored.

**Registers at call:**
AX = FEC6h
BL = new state (00h enabled, nonzero disabled)

**Return Registers:**
none

---

### INT 16h Function FED3h
### *UNDETERMINED FUNCTION*

*v7+ CPSCHED/DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FED3h
DS:SI -> 92-byte data record for *unknown*

**Return Registers:**
*unknown*

**Details:** This function is identical to Function FFD3h, and is implemented by the same code in DESKTOP.
**See also:** Function FFD3h

---

### INT 16h Function FEDCh
### *UNHOOK INTERRUPTS*

*v7+ CPSCHED*
*(undocumented)*

**Registers at call:**
AX = FEDCh

**Return Registers:**
AX,DX destroyed

---

### INT 16h Function FEEFh
### *INSTALLATION CHECK*

*v7+ CPSCHED/DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FEEFh
CX = 0000h

**Return Registers:**
CX = ABCDh if PC Tools scheduler (CPSCHED or DESKTOP) installed
BX = segment of resident portion
DX = (CPSCHED v8.0) resident CS

**Details:** This function is identical to Function FFD3h, and is implemented by the same code in DESKTOP.
**See also:** Function FFEFh

---

### INT 16h Function FEF1h
### *ALTERNATE INSTALLATION CHECK*

*v7 only CPSCHED/DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FEF1h
BX = *unknown*

**Return Registers:**
CX = 5555h if PC Tools scheduler (CPSCHED or DESKTOP) installed
DX = 5555h

**Details:** This function is identical to Function FFD3h, and is implemented by the same code in DESKTOP.
**See also:** Function FFF1h

**61-1 DESKTOP INT 16h functions available only when popped up:**

| *AX* | *Version* | *Description* |
|---|---|---|
| FF90h | 8.0+ | *undetermined* |
| FFA6h | 6.0+ | get address of *unknown value* into DS:SI |
| FFA8h | 6.0+ | *undetermined*, copies three strings from DS:SI to internal buffer |
| FFAAh | 6.0+ | *undetermined* |
| FFACh | 6.0+ | set *unknown value* to DL |

| | | |
|---|---|---|
| FFB3h | 5.5+ | *undetermined* |
| FFB4h | 5.5+ | set *unknown* flag (see Function FFBBh below) |
| FFBAh | 5.1+ | *undetermined*, returns unknown value in AX |
| FFBBh | 5.1+ | clear *unknown flag* (see Function FFB4h above) |
| FFBEh | 5.1+ | *undetermined* |
| FFBFh | 5.1+ | *undetermined*, outputs data to file handle BX |
| FFC0h | 5.1+ | *undetermined*, returns AX set to 0000h or FFFFh |
| FFC1h | 5.1+ | *undetermined* |
| FFC2h | 5.1+ | *undetermined* |
| FFC3h | 5.1+ | *undetermined* |
| FFC8h | 5.1+ | get *unknown pointer* into DS:SI |
| FFC9h | 5.1+ | copy CX bytes from DS:SI to clipboard, skipping bytes of 00h and 0Ah |
| FFCAh | 5.1+ | set *unknown value* to DX; destroys AX |
| FFCBh | 5.1+ | *select window parameters* for window DX |
| FFCDh | 5.1+ | *undetermined*, uses DS:DX as input |
| FFCFh | 5.1+ | close printer/print file |
| FFD0h | 5.1+ | *prepare to print* |
| FFD1h | 5.1+ | display print options menu, returns BX = copies and DX = destination |
| FFD2h | 5.1+ | *undetermined* |
| FFD5h | 5.1+ | *undetermined* |
| FFD6h | 5.1+ | *undetermined* |
| FFD7h | 5.1+ | *undetermined*, returns value in BL |
| FFD9h | 5.1+ | get *unknown value* into AX |
| FFDBh | 5.1+ | set *unknown value* to BL |
| FFDEh | 5.1+ | display popup menu for description pointed at by DS:DX |
| FFE0h | 5.1+ | *undetermined* |
| FFE2h | 5.1+ | *undetermined* |
| FFE3h | 5.1+ | print character in BL to currently-open printer or print file |
| FFE4h | 5.1+ | *undetermined* |
| FFE5h | 5.1+ | pop up file selection menu for filespec at DS:SI; returns AX=file handle |
| FFE6h | 5.1+ | check for and get keystroke into AX, calling INT 28h before checking |
| FFE7h | 5.1+ | *undetermined* |
| FFE9h | 5.1+ | *get file list* segment into BX |
| FFEAh | 5.1+ | display counted string at DS:SI |
| FFECh | 5.1+ | get key, calling INT 28h while awaiting keystroke |
| FFEDh | 5.1+ | get *unknown value* into AX |
| FFEEh | 5.1+ | define pulldown menus |
| FFF0h | 5.1+ | set *unknown value* to DX; destroys AX |
| FFF2h | 5.1+ | display help line pointed at by DS:SI |
| FFF4h | 5.1+ | *undetermined* |
| FFF6h | 5.1+ | invoke notepad editor (see entry below) |
| FFF7h | 5.1+ | *process menu bar entry* |
| FFF9h | 5.1+ | define screen refresh routine to be at ES:BX |
| FFFAh | 5.1+ | define standard pulldown menus ("Desktop" and "Window") |
| FFFBh | 5.1+ | process standard menu bar set with Function FFFAh |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

## INT 16h Functions FF91h and FF92h

v7+ DESKTOP
(undocumented)

### *UNDETERMINED FUNCTIONS*

**Registers at call:**
AX = FF91h
*others, if any, unknown*

**Return Registers:**
AX = 0000h

**Details:** These both call Function FFFDh after *unknown*; Function FF92h additionally sets an unknown item to 3 temporarily.

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

**See also:** Function FFFDh

---

## INT 16h Function FF93h

v7+ DESKTOP
(undocumented)

### *SET UNKNOWN FLAG*

**Registers at call:**
AX = FF93h

**Return Registers:**
none

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

## INT 16h Functions FF94h and FF95h

v7+ DESKTOP
(undocumented)

### *SET UNKNOWN VALUES*

**Registers at call:**
AX = function
    FF94h *unknown*
        CX = *unknown* (default 0017h)
    FF95h *unknown*
        BX = *unknown*

**Return Registers:**
none

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

## INT 16h Function FF96h

v7+ DESKTOP
(undocumented)

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = FF96h
CL = *unknown*

**Return Registers:**
AX = *unknown*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

## INT 16h Function FF97h

v7+ DESKTOP
(undocumented)

### *UNDETERMINED FUNCTION*

**Registers at call:**
AX = FF97h
DS:DX -> buffer for *unknown* (see Table 61-2)

**Return Registers:**
*unknown*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

**61-2 Format of buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 48 BYTEs | *unknown* |
| 30h | 128 BYTEs | *unknown* |

---

## INT 16h Function FF98h

v7+ DESKTOP
(undocumented)

### *OPEN \DESK.OVL FILE AND SEEK TO OVERLAY*

**Purpose:** Open the DESK.OVL file, seek to the specified offset, read in the overlay header, and seek to the offset specified by the header.

**Registers at call:**
AX = FF98h
DX = byte offset in file of overlay header

**Return Registers:**
BX = file handle for DESK.OVL file

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

**61-3 Format of overlay header:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | NUL-padded ASCII overlay filename |
| 0Ch | DWORD | offset within DESK.OVL file of actual overlay |

---

### INT 16h Function FF99h
### *UNDETERMINED FUNCTION*

*v7+ DESKTOP*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = FF99h | *unknown* |
| *others, if any, unknown* | |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FF9Ah
### *GET NAME OF COLOR SCHEME*

*v7+ DESKTOP*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = FF9Ah | ES:BX -> name of current color scheme |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FF9Bh
### *UNUSED*

*v7+ DESKTOP*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = FF9Bh | *unknown* |

**Details:** This call sounds a triple-length beep.

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FF9Eh
### *UNDETERMINED FUNCTION*

*v7+ DESKTOP*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = FF9Eh | *unknown* |
| DL = *unknown* | |
| bit 7: *unknown* | |
| bits 6-0: *function number* (00h,01h,other) | |
| *others, if any, unknown* | |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Functions FFA1h and FFA2h
### *UNDETERMINED FUNCTIONS*

*v7+ DESKTOP*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = function (FFA1h or FFA2h) | *unknown* |
| *others, if any, unknown* | |

**Details:** Function FFA1h is the same as Function FFA2h, except *unknown* is set to FFh. These functions call Functions FFC7h (remove window) and FFFDh.

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

**See also:** Functions FFA1h, FFC7h, and FFFDh

---

### INT 16h Function FFA3h, Subfunction 0000h
### *INSTALLATION CHECK*

*v7+ DATAMON*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = FFA3h | AX = segment of resident code |
| BX = 0000h | BX = 5555h if installed |
| CX = 0000h | CX = 5555h if installed |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

**See also:** Trusted Access INT 21h Function 3Fh (Chapter 66), Data Guardian INT 21h Function 4101h (Chapter 66)

---

### INT 16h Function FFA3h, Subfunctions 0001h to 0003h

### *GET UNKNOWN VALUE*

*v7+ DATAMON*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFA3h | |
| BX = CX = subfunction | |
| 0001h get *unknown value* | AX:BX -> *unknown*<br>CX = BX |
| 0002h get *unknown value* | AX = *unknown* (0 or 1)<br>CX = BX = AX |
| 0003h get *unknown value* | AX = *unknown* (0 or 1)<br>CX = BX = AX |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFA3h, Subfunctions 0004h and 0005h

### *SET/CLEAR UNKNOWN FLAG*

*v7+ DATAMON*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFA3h | none |
| BX = CX = 0004h set flag | |
| BX = CX = 0005h clear flag | |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFA3h, Subfunction 0006h

### *SET PSP SEGMENT*

*v7+ DATAMON*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFA3h | none |
| BX = CX = 0006h | |
| DX = *current PSP segment as known to DOS* or 0000h | |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFA4h

### *UNDETERMINED FUNCTION*

*v7+ DESKTOP*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFA4h | *unknown* |

**Details:** This function sets an unknown flag if *unknown* conditions are met.

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

**See also:** Function FEA4h

---

### INT 16h Function FFA7h

### *GET UNKNOWN PATH*

*v6.0+ DESKTOP*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFA7h | DS:SI -> ASCIZ path (*directory from which PCTools was run*) |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFA9h

### *GET VERSION STRING*

*v6.0+ DESKTOP*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFA9h | DS:SI -> version string |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFABh
### *GET EDITOR SETTINGS*

*v6.0+ DESKTOP (undocumented)*

**Registers at call:**
AX = FFABh

**Return Registers:**
DS:SI -> *editor setting strings*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFADh
### *SET UNKNOWN VALUE*

*v6.0+ DESKTOP (undocumented)*

**Registers at call:**
AX = FFADh
DL = *unknown*

**Return Registers:**
none

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFAEh
### *GET UNKNOWN VALUE*

*v6.0+ DESKTOP (undocumented)*

**Registers at call:**
AX = FFAEh

**Return Registers:**
AL = *unknown*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Functions FFAFh and FFB0h
### *SET UNKNOWN VALUES*

*v6.0+ DESKTOP (undocumented)*

**Registers at call:**
AX = function
    FFAFh set *unknown value*
        DL = *unknown*
    FFB0h set *unknown value*
        BL = *unknown*

**Return Registers:**
none

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFB1h
### *UNDETERMINED FUNCTION*

*v6.0+ DESKTOP (undocumented)*

**Registers at call:**
AX = FFB1h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFB2h
### *GET UNKNOWN VALUE*

*v5.5+ DESKTOP (undocumented)*

**Registers at call:**
AX = FFB2h

**Return Registers:**
DS:SI -> *unknown*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFB5h
### *GET/SET WINDOW PARAMETERS*

*v5.5+ DESKTOP (undocumented)*

**Registers at call:**
AX = FFB5h
BX = window specifier (000Fh to 0019h) (see Table 61-4)
DX = 0000h get, nonzero = set
ES:DI -> window parameter buffer (see Table 61-5)

**Return Registers:**
none

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** Function FFCBh

### 61-4 Values for window specifier:

| | |
|---|---|
| 000Fh | comm/FAX |
| 0014h | hotkey selection |
| 0015h | ASCII table |
| 0016h | system colors menu |

### 61-5 Format of window parameters:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | rows in window, not counting frame |
| 01h | BYTE | columns in window, not counting frame |
| 02h | BYTE | row number of top of window |
| 03h | BYTE | 2*column number of left of window |
| 04h | BYTE | character attribute for *unknown item* |
| 05h | BYTE | character attribute for background/border |
| 06h | BYTE | character attribute for *unknown item* |
| 07h | DWORD | pointer to *unknown item* on screen |
| 0Bh | 4 BYTEs | *unknown* |
| 0Fh | BYTE | nonzero if window may be resized |

**Note:** If running in monochrome mode, the character attributes at offsets 04h to 06h are stored unchanged, but attributes other than 07h, 0Fh, or 70h are changed to 07h on reading.

---

## INT 16h Function FFB6h
### *GET UNKNOWN VALUES*

*v5.5+ DESKTOP (undocumented)*

**Registers at call:**
AX = FFB6h

**Return Registers:**
AH = *unknown*
AL = *unknown*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

## INT 16h Function FFB7h
### *GET/SET UNKNOWN VALUE*

*v5.5+ DESKTOP (undocumented)*

**Registers at call:**
AX = FFB7h
BX = direction
  0000h copy to buffer
  else copy from buffer
DS:SI -> 70-byte buffer with *unknown*

**Return Registers:**
data copied

**Details:** Under v6.0+, this function is available only when popped up.

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

## INT 16h Function FFB8h
### *GET/SET UNKNOWN VALUES*

*v5.1+ DESKTOP (undocumented)*

**Registers at call:**
AX = FFB8h
BH = subfunction
  00h get

  nonzero set
    BL = new value for *unknown*
    CL = new value for *unknown* (v6.0+)
    CH = new value for

*unknown* (v6.0+)
DH = *unknown*

**Return Registers:**

BL = old value of *unknown*
CL = old value of *unknown* (v6.0+)
CH = old value of *unknown* (v6.0+)

AL = old value replaced by CL (v6.0+)
AH = old value replaced by CH (v6.0+)

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFB9h
### *UNDETERMINED FUNCTION*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFB9h
*others, if any, unknown*

**Return Registers:**
AX = *unknown*
CX = *unknown*
DS:SI -> *unknown*
ES:DI -> *unknown*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFBCh
### *RESTORE ORIGINAL SCREEN*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFBCh

**Return Registers:**
none

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFBDh
### *UNKNOWN DATABASE INDEXING MESSAGES*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFBDh
*others, if any, unknown*

**Return Registers:**
*unknown*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFC4h
### *GET UNKNOWN VALUES*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFC4h

**Return Registers:**
AL = *unknown*
BX = segment of *scratch space*
CX = segment of stored screen data (*section covered by window*)
DX = segment of window parameters for *unknown*
ES:BP -> *unknown*

**Details:** This function is available only when popped up in versions prior to 6.0.
**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFC5h
### *CHECK WHETHER DESKTOP LOADED RESIDENT*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFC5h

**Return Registers:**
BL = status
  nonzero if loaded resident
  00h if nonresident

**Details:** This function is available only when popped up; the application should call Function FFEFh first to ensure that DESKTOP is active.
**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** Functions FFEFh and FFF3h

### INT 16h Function FFC6h
### *SET UNKNOWN VALUE*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFC6h
BL = new value for *unknown*

**Return Registers:**
none

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFC7h
### *REMOVE WINDOW*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFC7h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFCCh
### *DISPLAY ASCIZ STRING CENTERED IN WINDOW*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFCCh
DS:SI -> ASCIZ string

**Return Registers:**
AX = *unknown*
CX = *unknown*
ES:DI -> address past last character displayed
  (v5.1/5.5)
  -> *unknown* on menu bar (v6.0)

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFCEh
### *SET UNKNOWN DELAYS*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFCEh
CX = *unknown*

**Return Registers:**
*none*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFD3h
### *UNDETERMINED FUNCTION*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFD3h
DS:SI -> 92-byte data record for *unknown*

**Return Registers:**
*unknown*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** Function FED3h

---

### INT 16h Function FFD4h
### *CREATE/OPEN/DELETE FILE*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFD4h
BH = operation
    3Ch create file (with no attributes)
    3Dh open file
    41h delete file
BL = access mode
    00h read only
    01h write only
    02h read/write
DS:SI -> ASCIZ filename

**Return Registers:**
BX = file handle
    0000h on error

**Details:** The operation is attempted in (in order) the directory *from which the desktop was started/run*, the directory specified with the filename, X:\PCTOOLS\, and X:\.
**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFD8h — *v5.1+ DESKTOP (undocumented)*
### *SAFE CREATE FILE*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFD8h | BX = file handle |
| DS:BX -> ASCIZ filename | 0000h on error |

**Details:** This call pops up a confirmation menu if the file already exists.

*This function is apparently only available when popped up.*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFDAh — *v5.1+ DESKTOP (undocumented)*
### *GET NAME OF LAST FILE OPENED*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFDAh | DS:SI -> filename |
| DS:SI -> *unknown* (v5.1/5.5 only) | |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFDCh — *v5.1+ DESKTOP (undocumented)*
### *UNHOOK*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFDCh | interrupt vectors 09h, 10h (v6.0+), 16h, 1Ch, and 21h restored to original values |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFDDh, Subfunction 0000h — *v5.1+ PCShell (undocumented)*
### *INSTALLATION CHECK*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFDDh | CX = 5555h |
| BX = 0000h | DX = 5555h if PCShell installed in resident mode |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFDDh, Subfunction 0001h — *v5.1+ PCShell (undocumented)*
### *REQUEST POP-UP*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFDDh | CF clear if request successful (PCShell will pop up) |
| BX = 0001h | CF set on error |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** Function FFDDh Subfunction 0003h

---

### INT 16h Function FFDDh, Subfunction 0002h — *v5.1-5.5 PCShell (undocumented)*
### *GET UNKNOWN VALUE*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFDDh | AL = result |
| BX = 0002h | 00h *unknown* |
| | 01h *unknown* |

**Details:** PCShell v6.0+ displays the error message "Incorrect PCRUN version", awaits a keystroke, and aborts the current process.
**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFDDh, Subfunction 0003h
### *REQUEST POP-UP*

*v5.1+ PCShell*
*(undocumented)*

**Registers at call:**
AX = FFDDh
BX = 0003h

**Return Registers:**
none

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** Function FFDDh Subfunction 0001h

---

### INT 16h Function FFDDh, Subfunction 0004h
### *GET UNKNOWN VALUE*

*v5.1+ PCShell*
*(undocumented)*

**Registers at call:**
AX = FFDDh
BX = 0004h

**Return Registers:**
CF clear if successful
DS:SI -> *unknown*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFDDh, Subfunctions 0005h and 0006h
### *UNDETERMINED FUNCTIONS*

*v5.1+ PCShell*
*(undocumented)*

**Registers at call:**
AX = FFDDh
BX = 0005h and 0006h
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** These functions reset various variables if certain conditions are met.
**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFDDh, Subfunctions 0007h and 0008h
### *SET/CLEAR UNKNOWN FLAG*

*v5.1+ PCShell*
*(undocumented)*

**Registers at call:**
AX = FFDDh
BX = subfunction
0007h set flag
0008h clear flag

**Return Registers:**
CF clear if successful (subfunction 0007h)
CF undefined (subfunction 0008h)

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFDDh, Subfunction 0009h
### *GET PCRUN PARAMETERS*

*v6.0+ PCShell*
*(undocumented)*

**Registers at call:**
AX = FFDDh
BX = 0009h

**Return Registers:**
CF clear if successful
DS:SI -> list of pointers (see Table 61-6)

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

**61-6 Format of returned pointer list:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | offset of WORD containing *unknown item* |
| 02h | WORD | offset of name of program to execute |
| 04h | WORD | offset of 80-byte buffer for *unknown item* |
| 06h | WORD | offset of buffer for *unknown item* (length in WORD preceding buffer) |
| 08h | WORD | offset of buffer for *unknown item* (length in WORD preceding buffer) |

---

### INT 16h Function FFDDh, Subfunction 000Ah
### *INSTALLATION CHECK*

*v6.0+ PCRUN*
*(undocumented)*

**Registers at call:**
AX = FFDDh
BX = 000Ah

**Return Registers:**
CX = 5555h if running
DX = 5555h

**Details:** This function also sets a flag.
**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFDDh, Subfunction 000Bh
### *UNDETERMINED FUNCTION*

*v6.0+ PCRUN*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFDDh | CX = 5555h if PCRUN active |
| BX = 000Bh | DX = 5555h |
| *others, if any, unknown* | |

**Details:** This function also clears the flag set by Function FFDDh Subfunction 000Ah.
**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFDFh
### *UNDETERMINED FUNCTION*

*v5.1+ DESKTOP*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFDFh | *unknown* |
| *others, if any, unknown* | |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFE1h
### *BEEP*

*v5.1+ DESKTOP*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFE1h | none |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFE8h
### *DISPLAY NUMBER*

*v5.1+ DESKTOP*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFE8h | DS:SI buffer filled in with alternating characters and attributes |
| CX = number | |
| DH = attribute | |
| DS:SI -> destination for ASCII number | |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFEBh
### *UNDETERMINED FUNCTION*

*v5.1+ DESKTOP*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFEBh | *unknown* |
| *others, if any, unknown* | |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFEFh
### *INSTALLATION CHECK*

*v5.1+ DESKTOP*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = FFEFh | CX = ABCDh if PC Tools DESKTOP.EXE installed |
| CX = 0000h | BX = segment of resident portion |
| | AX = *unknown* (v5.1/5.5 only) |

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** Functions FEEFh, FFC5h, and FFF3h

---

### INT 16h Function FFF1h
### *ALTERNATE INSTALLATION CHECK*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFF1h

BX = operation
 0000h leave *unknown* flag as is
 nonzero set *unknown* flag

**Return Registers:**
CX = 5555h if installed
DX = 5555h if installed

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFF3h
### *PREPARE TO UNLOAD RESIDENT DESKTOP*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFF3h

**Return Registers:**
none

**Details:** This function releases any EMS being used; restores the video mode, page, and cursor shape; and restores interrupt vectors.

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

**See also:** Functions FFC5h and FFEFh

---

### INT 16h Function FFF5h
### *GET SCREEN ATTRIBUTE ARRAY*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFF5h

**Return Registers:**
ES:BX -> screen attributes data structure (see Table 61-7)
AL = *unknown* (v6.0+)

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

**61-7 Format of attribute data structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -1 | BYTE | attribute for desktop background |
| 00h | BYTE | attribute for normal characters on desktop menu |
| 01h | BYTE | attribute for highlighted characters on desktop menu |
| 02h | 5 BYTEs | *unknown* |
| 07h | BYTE | attribute for dialog boxes |
| 08h | 15 BYTEs | *unknown* |
| 17h | BYTE | attribute for message boxes |

---

### INT 16h Function FFF8h
### *DRAW EMPTY WINDOW*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFF8h
DS:0000h -> window parameters structure (see Table 61-5)
DS:BX -> DWORD to store address of *unknown* on screen

**Return Registers:**
*unknown*

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

---

### INT 16h Function FFFCh
### *GET HOTKEYS AND KEYBOARD VECTOR*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFFCh

**Return Registers:**
ES:BX -> hotkey table (see Table 61-8)
DS:DX = original INT 09h vector

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)

**61-8 Format of hotkey table:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 2 BYTEs | scancode/shift state for desktop hotkey |
| 02h | 2 BYTEs | scancode/shift state for clipboard paste key |
| 04h | 2 BYTEs | scancode/shift state for clipboard copy key |
| 06h | 2 BYTEs | scancode/shift state for screen autodial key |

### INT 16h Function FFFDh
### *COPY UNKNOWN DATA*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = FFFDh

**Return Registers:**
AX destroyed

**Details:** This function copies 4000 bytes from *unknown* to *unknown* under certain circumstances.
**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** Function FF91h and FF92h

---

### INT 16h Functions FFFEh and FFFFh
### *SHOW/HIDE MOUSE CURSOR*

*v5.1+ DESKTOP*
*(undocumented)*

**Registers at call:**
AX = function
  FFFEh show cursor
  FFFFh hide cursor

**Return Registers:**
none

**Conflicts:** OPTIMA/ET-3000 Zoom TSR (Chapter 9), KBUF (Chapter 13)
**See also:** Mouse INT 33h Functions 0001h and 0002h (Chapter 21)

---

### INT 2Fh Function 6284h, Subfunction 0000h
### *INSTALLATION CHECK*

*v7.0+ DATAMON*
*(undocumented)*

**Registers at call:**
AX = 6284h
BX = 0000h
CX = 0000h

**Return Registers:**
AX = segment of resident code
BX = 5555h
CX = 5555h

---

### INT 2Fh Function 6284h, Subfunction 0001h
### *GET UNKNOWN VALUE*

*v7.0+ DATAMON*
*(undocumented)*

**Registers at call:**
AX = 6284h
BX = 0001h
CX = 0001h

**Return Registers:**
AX:BX -> *unknown*
CX = BX

---

### INT 2Fh Function 6284h, Subfunction 0002h
### *GET UNKNOWN VALUE*

*v7.0+ DATAMON*
*(undocumented)*

**Registers at call:**
AX = 6284h
BX = 0002h
CX = 0002h

**Return Registers:**
AX = *unknown*
BX = *unknown*
CX = AX
DX = BX

---

### INT 2Fh Function 6284h, Subfunction 0003h
### *SET UNKNOWN FLAGS*

*v7.0+ DATAMON*
*(undocumented)*

**Registers at call:**
AX = 6284h
BX = 0003h
CX = flags
  bit 12: *unknown*
  bit 10: *unknown*
  bit 5: *unknown*
  bit 3: *unknown*
DX = flags
  bit 15: *unknown*

**Return Registers:**
none

---

## INT 2Fh Function 6284h, Subfunction 0004h

*v8.0+ DATAMON*
*(undocumented)*

### *UNDETERMINED FUNCTION*

| Registers at call: | Return Registers: |
|---|---|
| AX = 6284h | AX = 5555h |
| BX = 0004h | BX = *unknown* (0800h) |
| CX = 0004h | CX = *unknown* (FCCCh) |

## CHAPTER 62

# *Norton Utilities*

The Norton Utilities package is known, for good reason, as "the granddaddy of DOS disk utilities." From its somewhat shaky start as the output of Peter Norton's one-man company during the first year of the IBM PC's existence, when its only claim to fame was the ability to recover an erased file, this package created the entire disk-utility market niche and is now a mainstay of the product line at Symantec Corp., in Cupertino, CA.

Version 6.01 includes not only UNERASE, which recovers files, but also the Norton Disk Doctor and the Norton Disk Editor. For files you want to erase permanently, there's WIPEINFO, and the Norton Speedisk program safely and speedily defragments your hard disk drives. A variety of other utilities warehouse deleted files, encrypt files on the fly, prevent access to a disk, repair damaged database and spreadsheet files, test and optimize the disk interleave, enhance batch files, set your cursor size, and more.

This chapter lists not only those interrupts used by the NU programs themselves, but also by other utilities produced over the years under the Norton name.

---

### INT 16h Function F398h
### *INSTALLATION CHECK*

*NORTON GUIDES*
*(undocumented)*

**Registers at call:**
AX = F398h

**Return Registers:**
AX = 6A73h ("js")
BH = scan code of current hot key
BL = ASCII code of current hot key

**Details:** NG.EXE was written by John Socha.
**Conflicts:** Compaq BIOS (Chapter 7)

---

### INT 2Fh Function FE00h
### *INSTALLATION CHECK/STATUS REPORT*

*NORTON UTILITIES 5.0+ TSRs*
*(undocumented)*

**Registers at call:**
AX = FE00h
DI = 4E55h ("NU")
SI = TSR identifier
    4346h ("CF") NCACHE-F (v5), NCACHE (v6+)
    4353h ("CS") NCACHE-S (v5 only)
    4443h ("DC") DISKREET
    444Dh ("DM") DISKMON
    4653h ("FS") FILESAVE (v5) / EP (v6.0+)

**Return Registers:**
SI = TSR reply: lowercase version of SI on entry (SI OR 2020h)
AH = status
    00h installed but disabled internally
    01h installed and enabled
AL = status
    00h NCACHE-x or DISKREET installed
    01h FILESAVE / EP / DISKMON v6 installed
    45h DISKMON v5 installed
BX = length of *.INI file (DISKMON and FILESAVE/EP only) (see Table 62-1)
CX = segment of resident portion

FFFFh if completely loaded high (NCACHE)
---**FILESAVE/EP**---
DL = *unknown* (apparently always 00h)
---**DISKMON**---
DX = *unknown* (apparently always 1AE6h [v5] / 1B86h [v6])

**Details:** The value returned in CX is incorrect for NCACHE 6.00.
**Conflicts:** none, but see Table 2-2
**See also:** Function FE01h, FE02h, FE03h, FE04h, and FE05h

**62-1 Format of DISKMON.INI file:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -6Ch | 108 BYTEs | (in memory copy only) |
| | | list of filenames which are always protected: IBMBIO.COM/IBMDOS.COM, IO.SYS/MSDOS.SYS, TBIOS.SYS/TDOS.SYS, MIO.SYS/IO.BIN, COMMAND.COM |
| 00h | BYTE | *unknown*, always 01h |
| 01h | BYTE | disk light (00h off, 01h on) |
| 02h | BYTE | disk protection (00h off, 01h on) |
| 03h | BYTE | protected areas |
| | | 01h system area |
| | | 02h files |
| | | 03h system area and files |
| | | 04h entire disk |
| 04h | BYTE | floppy access (00h not allowed, 01h allowed) |
| 05h | 27 BYTEs | filename extension list (9 entries) (lowercase, blank padded or = 000000h) |
| 20h | 240 BYTEs | filename list (20 entries) (lowercase, name and extension blank padded, with '.' |

**Note:** CX:0508h -> copy in installed TSR for v5; CX:052Fh -> copy in installed TSR for v6

**62-2 Format of FILESAVE.INI / EP.INI file:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 26 BITs | drive list (bit set: file protection on, cleared: off): |
| 00h | BYTE | drives A: - H: |
| 01h | BYTE | drives I: - P: |
| 02h | BYTE | drives Q: - X: |
| 03h | BYTE | drives Y: - Z: |
| 04h | BYTE | which files to protect |
| | | 00h all files |
| | | 01h all files with extension in list |
| | | 02h all files except those with extension in list |
| 05h | 27 BYTEs | filename extension list (9 entries, uppercase, ASCIZ) |
| 20h | BYTE | include files with archive bit clear (00h no, 01h yes) |
| 21h | WORD | number of days after which files are purged (0 = never) |
| 23h | WORD | max kilobytes of erased file space to hold (0 = all) |

**Note:** CX:03D2h -> copy in installed TSR for v5.0; CX:03F5h -> copy in installed TSR for v6.0

### INT 2Fh Function FE01h
### *ENABLE*

*NORTON UTILITIES 5.0+ TSRs*
*(undocumented)*

**Registers at call:**
AX = FE01h
DI = 4E55h ("NU")
SI = TSR identifier (see Function FE00h)

**Return Registers:**
SI = TSR reply (lowercase version of entry SI, i.e. SI OR 2020h)
AX = status
0002h successful (DISKMON, FILESAVE, EP)
FE00h successful (NCACHE-x, DISKREET)

**Details:** If the enable/disable calls are used on DISKMON or NCACHE-x, the status report generated by the programs still indicates the previous state, and DISKMON.INI is not updated.

This call apparently has no effect on DISKREET.

**Conflicts:** none, but see Table 2-2

**See also:** Functions FE00h and FE02h

---

### INT 2Fh Function FE02h
### *DISABLE*

*NORTON UTILITIES 5.0+ TSRs (undocumented)*

**Registers at call:**
AX = FE02h
DI = 4E55h ("NU")
SI = TSR identifier (see Function FE00h)

**Return Registers:**
SI = TSR reply (lowercase version of entry SI, i.e. SI OR 2020h)
AX = status
0004h successful (DISKMON, FILESAVE)
FE00h successful (NCACHE-x, DISKREET)

**Details:** (see also Function FE01h).

This function appears to be unsafe, as the cache buffers are not flushed.

**Conflicts:** none, but see Table 2-2

**See also:** Functions FE00h and FE01h

---

### INT 2Fh Function FE03h
### *FLUSH BUFFERS*

*NORTON UTILITIES 5.0+ TSRs (undocumented)*

**Registers at call:**
AX = FE03h
DI = 4E55h ("NU")
SI = TSR identifier (see Function FE00h)

**Return Registers:**
SI = TSR reply (lowercase version of entry SI, i.e. SI OR 2020h)
AX = status
0006h *successful*

**Details:** This function is only supported by DISKMON, FILESAVE, and NCACHE-x. It is useful for flushing NCACHE before rebooting.

**Conflicts:** none, but see Table 2-2

**See also:** Functions FE00h and FE10h

---

### INT 2Fh Functions FE04h and FE05h
### *UNDETERMINED FUNCTION*

*NORTON UTIL 5.0+ DISKMON, FILESAVE / EP (undocumented)*

**Registers at call:**
AX = function (FE04h and FE05h)
DI = 4E55h ("NU")
SI = TSR identifier (see Function FE00h)

**Return Registers:**
SI = TSR reply (lowercase version of entry SI, i.e. SI or 2020h)
AX = status
0008h *successful* (Function FE04h)
000Ah *successful* (Function FE05h)

**Details:** Function FE05h is reportedly dangerous.

**Conflicts:** none, but see Table 2-2

**See also:** Function FE00h

---

### INT 2Fh Function FE10h
### *REBOOT*

*NORTON UTILITIES 6.0 NCACHE (undocumented)*

**Registers at call:**
AX = FE10h
DI = 4E55h ("NU")
SI = TSR identifier (see Function FE00h)

**Return Registers:**
SI = TSR reply (lowercase version of entry SI, i.e. SI or 2020h)
AX = status

**Details:** This function is probably used to flush NCACHE buffers and reboot when Ctrl-Alt-Del is detected.

**Conflicts:** none, but see Table 2-2

**See also:** Function FE03h

## CHAPTER 63

# *Miscellaneous Quarterdeck Programs*

Quarterdeck Office Systems produces a number of programs in addition to their flagship products DESQview and QEMM-386. This chapter describes the interrupts used by DOS-UP.SYS, HOOKROM.SYS, HIMEMQ, QRAM, MANIFEST, DVDOS4GX.DVR, VIDRAM, and QDPMI.SYS.

---

### INT 21h Function 2B01h
### *INSTALLATION CHECK*

*DOS-UP.SYS v2.00*
*(undocumented)*

**Registers at call:**
AX = 2B01h
CX = 444Dh ('DM')
DX = 4158h ('AX')

**Return Registers:**
AX = 0000h if installed
BX = version??? (0002h)
CX = 4845h ('HE')
DX = 5245h ('RE')
ES = DOS-UP driver segment

---

### INT 21h Function 4402h
### *HOOKROM.SYS - GET HOOKED VECTOR TABLE*

*Quarterdeck*
*(undocumented)*

**Registers at call:**
AX = 4402h
BX = file handle for device "HOOKROM$"
CX = 0004h
DS:DX -> DWORD buffer for address of hooked vector table (see Table 63-1)

**Return Registers:**
CF clear if successful
DS:DX buffer filled
CF set on error
AX = error code (01h,05h,06h,0Dh) (see Table 24-2)

**See also:** Memory Manager Function 4402h Subfunction 01h (Chapter 37)

**63-1 Format of hooked vector table entry:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 5 BYTEs | FAR jump to actual interrupt handler (end of table if first byte is not EAh) |
| 05h | BYTE | interrupt vector number |

---

### INT 2Fh Function D200h
### *INSTALLATION CHECK*

*Quarterdeck RPCI*

**Registers at call:**
AX = D200h
BX = 5144h ("QD")
CX = 4D45h ("ME")
DX = 4D30h ("M0")

**Return Registers:**
AL = FFh installed
if BX,CX,DX registers were as specified on entry:
BX = 4D45h ("ME")
CX = 4D44h ("MD")
DX = 5652h ("VR")

**Details:** AH=D2h is the default multiplex number for the Quarterdeck RPCI (Resident Program Communication Interface), but it may use the first available multiplex number in the range D2h through FFh, then C0h through D1h.

The RPCI is supported by QEMM v5.0+, QRAM, MANIFEST, VIDRAM, etc.

For AL <> 0, if the BX/CX/DX values don't match the identifier of a Quarterdeck product, it just chains to the previous INT 2Fh handler.

**Conflicts:** PCL-838 (Chapter 66); see also Table 2-2

**See also:** Function D201h Subfunctions 4849h, 4D41h, 4D45h, and 5649h; QEMM Function D201h Subfunction 5145h (Chapter 39)

---

### INT 2Fh Function D201h, Subfunction 4849h

*Quarterdeck RPCI*

***GET QD HIMEM PRESENCE***

**Registers at call:**
AX = D201h
BX = 4849h ("HI")
CX = 4D45h ("ME")
DX = 4D51h ("MQ")

**Return Registers:**
BX = 4F4Bh ("OK")
ES:DI -> HIMEM entry point

**Details:** AH=D2h is the default multiplex number, but any value in the range C0h-FFh may be used (see Function D200h for details). **Conflicts:** PCL-838 (Chapter 66); see also Table 2-2

**See also:** Function D200h, QEMM Function D201h Subfunction 5145h (Chapter 39)

---

### INT 2Fh Function D201h, Subfunction 4849h

*Quarterdeck RPCI*

***QEMM/QRAM v5.0+ - GET HIRAM MEMORY CHAIN***

**Registers at call:**
AX = D201h
BX = 4849h ("HI")
CX = 5241h ("RA")
DX = 4D30h ("M0")

**Return Registers:**
BX = 4F4Bh ("OK")
CX = segment of start of HIRAM chain
DX = reserved block owner (QEMM/QRAM code segment)

**Details:** AH=D2h is the default multiplex number, but any value in the range C0h-FFh may be used (see Function D200h for details).

The HIRAM memory chain has the same format as the regular DOS 4.0 memory chain (see INT 21h Function 52h), except that XMS Upper Memory Blocks have the block header program name field set to "UMB"; blocks whose "owner" field is set to the reserved segment returned in DX are locked-out regions such as video memory and ROMs.

**Conflicts:** PCL-838 (Chapter 66); see also Table 2-2

**See also:** Function D200h, DR-DOS Function 12FFh Subfunction 0006h (Chapter 26)

---

### INT 2Fh Function D201h, Subfunction 4D41h

*Quarterdeck RPCI (undocumented)*

***MANIFEST v1.0+ - INSTALLATION CHECK***

**Registers at call:**
AX = D201h
BX = 4D41h ("MA")
CX = 4E49h ("NI")
DX = 4645h ("FE")

**Return Registers:**
BX = 5354h ("ST")

**Conflicts:** PCL-838 (Chapter 66); see also Table 2-2

**See also:** Function D200h

---

### INT 2Fh Function D201h, Subfunction 4D45h

*Quarterdeck RPCI (undocumented)*

***DVDOS4GX.DVR*** *- UNDETERMINED FUNCTION*

**Registers at call:**
AX = D201h
BX = 4D45h ("ME")
CX = 5155h ("QU")
DX = 5044h ("PD")

**Return Registers:**
AL = FFh
BX = 4F4Bh ("OK")

**Details:** This function is called by QEMM 6.03. It performs a variety of actions before setting return registers. **Conflicts:** PCL-838 (Chapter 66); see also Table 2-2. **See also:** Function D200h

### INT 2Fh Function D201h, Subfunction 5649h
### *VIDRAM v5.0+ - INSTALLATION CHECK*

*Quarterdeck RPCI*
*(partially doc.)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = D201h | BX = 4F4Bh ("OK") |
| BX = 5649h ("VI") | ES:DI -> VIDRAM entry point |
| CX = 4452h ("DR") | |
| DX = 414dh ("AM") | |

**Conflicts:** PCL-838 (Chapter 66); see also Table 2-2
**See also:** Function D200h

#### 63-2 VIDRAM entry point calling convention:

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 00h get status | CF set on error<br>CF clear if successful<br>AL = VIDRAM state (see return registers)<br>BL = extra RAM status<br>00h VIDRAM does not use extra RAM<br>01h VIDRAM uses EMS as extra RAM<br>02h VIDRAM uses EGA as extra RAM<br>BH = feature flags (see Table 63-3)<br>CL = current monitor (01h = mono, 80h = color)<br>SI = current top of memory (paragraph)<br>DI = segment of start of HiRAM chain |

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 01h setup<br>AL = VIDRAM state (00h off, 01h no EGA graphics, 02h no graph)<br>BL = extra RAM status (see at left)<br>BH = feature flags (see Table 63-3)<br>CL = monitor (01h = monochrome, 80h = color)<br>SI = new top of memory (paragraph)<br>DI = segment of start of HiRAM chain | CF set on error<br>CF clear if successful |

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 02h get end address of VIDRAM code | CF set on error<br>CF clear if successful<br>ES:DI -> VIDRAM partial map context (see Tables 63-4 and 63-5) |

#### 63-3 Bitfields for VIDRAM feature flags:

| | |
|---|---|
| bit 0 | override enabled |
| bit 1 | mapped memory detected in A000h-B000h range |
| bit 2 | top of memory not at 640K |
| bit 3 | MDA detected |
| bit 4 | high RAM exists in video area |
| bit 5 | mapped memory detected in video area |
| bits 6-7 | *reserved* |

#### 63-4 Format of partial map context (EMS 3.2):

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | EMS version ID (32h) |
| 01h | WORD | EMM handle for this entry |
| 03h | BYTE | number of frames |
| 04h | BYTE | first page frame |
| 05h | WORD | offset from ES to previously saved map |

### 63-5 Format of partial map context (EMS 4.0):

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | EMS version ID (40h) |
| 01h | WORD | mappable segment count |
| 03h | N WORD | mappable segments |
| | WORD | *offset to previously saved map* |

## INT 2Fh Function DE01h, Subfunction 4450h

### *INSTALLATION CHECK*

*Quarterdeck QDPMI.SYS v1.0*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DE01h | AL = FFh if installed |
| BX = 4450h ("DP") | BX = 4D42h ("MB") |
| CX = 4D49h ("MI") | CX = 4921h ("I!") |
| DX = 3039h ("09") | DX = 8F4Fh |
| | ES:DI -> filename of DPMI host overlay |

**Details:** The installation check consists of testing for the existence of the character device QDPMI$$$.

**Conflicts:** none, but see Table 2-2

**See also:** DPMI INT 2Fh Function 1687h (Chapter 30), DPMI INT 31h Function 0000h (Chapter 30)

# CHAPTER 64

# *FAX Software*

FAX software is, at the time of writing, a most unstandardized segment of the industry, although the actual rules for fax transmission as established by the CCITT (Comite Consultatif International Telegraphique et Telephonique, the international governing body on communications standards), are quite strict.

At least four different standard interfaces to the PC architecture exist, however, and none is compatible with any of the others. Three of the four (Class 1, Class 2, and FaxBIOS) are described in this chapter, while the fourth (CAS) is covered by Chapter 65.

## BitFax Scheduler

BitFax is a product of BIT Software, Cambridge, MA, and is bundled with many fax/modem cards.

The installation check consists of checking for the signature " BitFax Scheduler" beginning two bytes past the INT 66h handler. There are many potential conflicts with other software using interrupts in the range 60h to 67h, which are listed in Table 2-3.

---

### INT 66h Functions 01h and 02h

*BitFax Scheduler*

***SET MODES***

**Registers at call:**
AH = function
  01h *set unknown mode*
  02h *set unknown mode*

**Return Registers:**
none

---

### INT 66h Function 03h

*BitFax Scheduler*

***SCHEDULE FAX TRANSMISSIONS***

**Registers at call:**
AH = 03h
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 05h

---

### INT 66h Function 04h

*BitFax Scheduler*

***GET STATUS***

**Registers at call:**
AH = 04h

**Return Registers:**
AX = *unknown* (0000h or 0001h)
DX = *BitSched version* (for versions >= 3.00)
  9796h (v3.00)
  97E6h (v3.02)
  92D0h (v3.04.06)
  9510h (v3.06.02)

**See also:** Functions 06h and 3345h, FaxBIOS INT 2Fh Function 8000h (below)

---

**INT 66h Function 05h** *BitFax Scheduler*

### *CONVERT FILE AND SEND FAX*

**Registers at call:**
AH = 05h
BX:CX -> command block (see Table 64-1)
*others, if any, unknown*

**Return Registers:**
*unknown*

**See also:** Function 03h

**64-1 Format of command block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 18 BYTEs | *configuration bytes* |
| 12h | BYTEs | ASCIZ temporary file name to place converted fax |
| 52h | BYTEs | ASCIZ directory containing BitFax executables |
| 92h | BYTEs | ASCIZ telephone number |
| C2h | BYTE | cover page flag (00h don't send, 01h send cover page) |
| C3h | 15 BYTEs | *configuration bytes* |
| E2h | BYTEs | ASCIZ path of BITFAX.TRA file (*containing additional configuration information*) |
| 122h | BYTEs | *configuration bytes* |
| 12Ch | BYTE | 00h don't send cover page<br>01h send cover page |
| 12Dh | 7 BYTEs | *configuration bytes* |
| 134h | BYTEs | ASCIZ path of file to send |
| 174h | BYTEs | *more configuration bytes* |
|  |  | *additional fields, if any, unknown* |

---

**INT 66h Function 06h** *BitFax Scheduler*

### *SET MODE*

**Registers at call:**
AH = 06h

**Return Registers:**
DX = *BitSched version* (same as Function 04h)

**See also:** Function 04h

---

**INT 66h Function 3345h** *BitFax Scheduler*

### *REMOVE TSR FROM MEMORY*

**Registers at call:**
AX = 3345h

**Return Registers:**
AX = FFFFh error removing TSR

**See also:** Function 04h, Gammafax INT 2Fh Function 2Ah (below), Communicating Applications Spec INT 2Fh Function CB00h (Chapter 65)

## FaxBIOS

The FaxBios 1.0 applications programming interface (API) standard proposed by the FaxBios Association in October 1991 is intended to provide fax applications with a standard set of functions to utilize fax modems for receiving and sending faxes and to enable fax applications to cooperate.

The FaxBios API facilities include MS-DOS and Windows 3.0 interfaces, two methods of status reporting, phone directory and logging operations, reading and writing of graphics files, and drivers for call placement and file transmission, including the ability to schedule and transmit faxes in the background.

FaxBIOS may use any multiplex number from 80h through FFh; to determine whether it is installed, it is necessary to poll all multiplex numbers for one which returns the signature shown in the installation check below. As with all programs using multiplex numbers in this range, there are multiple potential conflicts which are listed in Table 2-2.

---

### INT 2Fh Function 8000h
*FaxBIOS interface*

### *INSTALLATION CHECK*

**Registers at call:**
AX = 8000h
DX = 0000h
DI = 0000h

**Return Registers:**
AL = FFh if installed on specified multiplex number
DX:DI -> signature " FaxBiosjpc"

**See also:** Function 80FBh, Gammafax Function 2Ah (below)

---

### INT 2Fh Function 80FBh
*FaxBIOS interface*

### *COMMAND SUBMISSION*

**Registers at call:**
AX = 80FBh
BX = function number (see Table 64-2)
DX:DI -> command buffer (see Table 64-4)

**Return Registers:**
AL = FFh if submitted OK
CX = result code (see Table 64-3)

**See also:** Function 8000h, Comwave Function CBDDh (below)

**64-2 Values for function number:**

| | |
|---|---|
| 0001h | SYS_LOGIN |
| 0002h | SYS_LOGOUT |
| 0003h | SYS_GET_FAXAPP_INFO |
| 0004h | STAT_IO_GET |
| 0005h | STAT_FAXBIOS_GET |
| 0006h | PDIR_OPEN |
| 0007h | PDIR_CLOSE |
| 0008h | PDIR_READ_PERSON |
| 0009h | PDIR_PARTIAL_READ |
| 000Ah | PDIR_READ_GROUP |
| 000Bh | PDIR_READ_MEMBER_LIST |
| 000Ch | PDIR_WRITE_PERSON |
| 000Dh | PDIR_WRITE_GROUP |
| 000Eh | PDIR_DELETE_PERSON |
| 000Fh | PDIR_DELETE_GROUP |
| 0010h | PDIR_READ_GROUP_LIST |
| 0011h | PDIR_IN_GROUP |
| 0012h | PDIR_OUT_GROUP |
| 0013h | SCHED_OPEN |
| 0014h | SCHED_ADD_DEST |
| 0015h | SCHED_ADD_FILE |
| 0016h | SCHED_SET_PARAMS |
| 0017h | SCHED_CANCEL |
| 0018h | SCHED_CLOSE |
| 0019h | SLOG_OPEN |
| 001Ah | SLOG_CLOSE |
| 001Bh | SLOG_SHORT_ENV_STAT |
| 001Ch | SLOG_LONG_ENV_STAT |
| 001Dh | SLOG_DEST_STAT |
| 001Eh | SLOG_FILE_STAT |
| 001Fh | SLOG_CANCEL_ENV |

| | |
|---|---|
| 0020h | RLOG_OPEN |
| 0021h | RLOG_CLOSE |
| 0022h | RLOG_READ |
| 0023h | GRAPH_GET_FILE_TYPE |
| 0024h | GRAPH_EXPORT_FILE |
| 0025h | GRAPH_GET_LAYOUT_INFO |
| 0026h | GRAPH_CREATE_FILE |
| 0027h | GRAPH_CLOSE_FILE |
| 0028h | GRAPH_CREATE_PAGE |
| 0029h | GRAPH_WRITE |
| 002Ah | GRAPH_END_PAGE |
| 002Bh | GRAPH_OPEN_FILE |
| 002Ch | GRAPH_GOTO_PAGE |
| 002Dh | GRAPH_READ |
| 002Eh | IOCTL_GET |
| 002Fh | IOCTL_SET |
| 0030h | IOCTL_ANSWER_FAX |
| 0031h | IOCTL_DIAL |

**64-3 Values for result code:**

| | |
|---|---|
| 0000h | successful |
| 0001h | not prepared or servicing another client (busy) |
| 0002h | call failed due to sharing (LOCKED) |
| 0003h | logged-in client limit reached (FULL) |
| 0004h | transport denied |
| 0005h | not implemented |
| 0006h | aborted while in progress (ABORTED) |
| 0007h | permissions denied |
| 0008h | requested data is no longer valid |
| 0080h | unspecified system error occurred |
| 0081h | an internal file was not found |
| 0082h | an internal file could not be created |
| 0083h | an internal file could not be opened |
| 0084h | an internal file could not be closed |
| 0085h | error occurred writing to an internal file |
| 0086h | error occurred reading from an internal file |
| 0087h | bad or corrupted file encountered |
| 0088h | an access violation occurred |
| 0089h | an internal file is empty |
| 008Ah | insufficient memory to process request |
| 008Bh | FaxBIOS was unable to issue a handle |
| 008Ch | an error internal to FaxBIOS occurred |
| 008Dh | no room on disk |
| 0100h | unspecified error accessing client file |
| 0101h | file not found |
| 0102h | creation fault |
| 0103h | open fault |
| 0104h | close fault |
| 0105h | write fault |
| 0106h | read fault |

## INT 2Fh Function 80FBh

| | |
|---|---|
| 0107h | file corrupted |
| 0108h | access violation |
| 0109h | empty file |
| 0200h | unspecified argument error |
| 0201h | bad function |
| 0202h | bad option |
| 0203h | bad structure size |
| 0204h | bad buffer size |
| 0205h | bad client ID |
| 0300h | unspecified error with token |
| 0301h | cover sheet token was invalid |
| 0302h | logo token was invalid |
| 0303h | signature token was invalid |
| 0304h | font token was invalid |
| 0305h | phone directory token was invalid |
| 0306h | outbound route token was invalid |
| 0307h | priority token was invalid |
| 0308h | sort token was invalid |
| 0309h | billing token was invalid |
| 0400h | unspecified handle error |
| 0401h | bad Phone Directory handle |
| 0402h | bad scheduling handle |
| 0403h | bad read send log handle |
| 0404h | bad read receive log handle |
| 0405h | bad graphics handle |
| 0500h | data passed in structure was invalid |
| 0501h | name field given is invalid |
| 0502h | phone number given is invalid |
| 0503h | poll code submitted is invalid |
| 0504h | file type constant was invalid |
| 0505h | BFT constant not defined or supported |
| 0506h | resolution not defined or supported |
| 0507h | page length not defined or supported |
| 0508h | page width not defined or supported |
| 0509h | date & time requested are ridiculous |
| 050Ah | Subject text was not an ASCIIZ string |
| 050Bh | From text was not an ASCIIZ string |
| 050Ch | requested envelope ID was not found |
| 050Dh | requested envelope ID is not valid |
| 050Eh | envelope requested was not found |
| 050Fh | destination index is out of range |
| 0510h | file index is out of range |
| 0511h | index into receive log is out of range |
| 0512h | file name specified was incomplete or invalid |
| 0513h | page selected was out of range |
| 0514h | bit width more than byte width |
| 0515h | mode for open is not defined |
| 0516h | person index is out of range |
| 0517h | person ID is out of range |

| | |
|---|---|
| 0518h | group index out of range or invalid |
| 0519h | group ID out of range or invalid |
| 051Ah | range of indices to read is invalid |
| 051Bh | group name given is invalid |
| 051Ch | field_to_use is badly specified |
| 051Dh | predicate invalid for field specified |
| 0600h | unspecified client procedure error |
| 0601h | device of interest is not present |
| 0602h | device of interest has been removed |
| 0603h | device of interest is not responding |
| 0604h | device of interest is disabled |
| 0605h | could not dial because device was in use |
| 0606h | maximum destination limit exceeded |
| 0607h | maximum file limit exceeded |
| 0608h | scheduling closed with no destination |
| 0609h | scheduling closed with no files or poll |
| 060Ah | scheduling closed with no parameters specified |
| 060Bh | file type specified does not match file |
| 060Ch | file type specified is not supported |
| 060Dh | file submitted is not exportable |
| 060Eh | file type specified is not imageable |
| 060Fh | error converting file |
| 0610h | envelope could not be cancelled |
| 0611h | Phone Directory is full |
| 0612h | record is already in the Phone Directory |
| 0613h | selected group in Phone Directory is full |
| 0614h | person is already in the group |
| 0615h | person is not in the group & cannot be removed |
| 0616h | a graphics file to be created already exists |
| 0617h | a grphics file to be read is empty |
| 0618h | GRAPH_CREATE_PAGE called before GRAPH_END_PAGE |
| 0619h | graph read or write attempted without goto or create |
| 061Ah | graph page contains no data |
| 061Bh | Phone Directory is already open for this client |
| 061Ch | schedule log is already open for this client |
| 061Dh | receive log is aready open for this client |
| 061Eh | Phone Directory function requires write mode |
| 0800h | denied exclusive use of the API |

**64-4 Format of SYS_LOGIN command buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | structure size |
| 02h | WORD | function number |
| 04h | WORD | return code |
| 06h | WORD | client ID |
| 08h | WORD | API Major Version |
| 0Ah | WORD | API Minor Version |
| 0Ch | DWORD | reserved for manufacturer's use |
| 10h | 22 BYTEs | manufacturer's ID |
| 26h | WORD | highest possible device number |

| | | |
|---|---|---|
| 28h | WORD | maximum destinations per envelope |
| 2Ah | WORD | maximum files per envelope |
| 2Ch | WORD | FaxBIOS capabilities (see Table 64-5) |
| 2Eh | DWORD | T.30 capabilities (see Table 64-6) |
| 32h | WORD | IPC handle |
| 34h | DWORD | amount of memory needed to load |
| 38h | WORD | scope (00h for public, nonzero for private) |
| 3Ah | 6 BYTEs | future expansion |
| 40h | WORD | structure size |

### 64-5 Bitfields for FaxBIOS capabilities:

| | |
|---|---|
| bit 0 | transmit supported |
| bit 1 | receive supported |
| bit 2 | IOCTL supported |
| bit 3 | IOCTL_DIAL supported |
| bit 4 | IOCTL_ANSWER_FAX supported |
| bit 5 | manual transmit supported |
| bit 6 | optional phone services supported |
| bit 7 | canonical phone objects |
| bit 8 | seam with next supported |

### 64-6 Bitfields for T.30 capabilities:

| | |
|---|---|
| bit 0 | low vertical resolution (minimum) |
| bit 1 | high vertical resolution |
| bit 2 | page width 107mm (4.21 in) |
| bit 3 | page width 151mm (5.91 in) |
| bit 4 | page width 215mm (8.46 in) (minimum) |
| bit 5 | page width 255mm (10.04 in) |
| bit 6 | page width 303mm (11.93 in) |
| bit 7 | unused |
| bit 8 | page length 297mm (11.69 in) (minimum) |
| bit 9 | page length 364mm (14.33 in) |
| bit 10 | page length 279mm (11 in) |
| bit 11 | page length unlimited |
| bit 12 | Group 4 resolution 300x300 |
| bit 13 | Group 4 resolution 400x400 |
| bit 14 | able to respond to poll from remote |
| bit 15 | able to poll remote |
| bit 16 | binary file transfer supported |

### 64-7 Format of SYS_LOGOUT command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | structure size |
| 02h | WORD | function number |
| 04h | WORD | return code |
| 06h | WORD | client ID |
| 08h | DWORD | client tag (for client's use) |
| 0Ch | 6 BYTEs | future expansion |
| 12h | WORD | structure size |

### 64-8 Format of SYS_GET_FAXAPP_INFO command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | 80 BYTEs | FaxBIOS data |
| 5Ch | 80 BYTEs | default cover |

| | | |
|---|---|---|
| ACh | 80 BYTEs | default logo |
| FCh | 80 BYTEs | default signature |
| 14Ch | 80 BYTEs | default font 10 |
| 19Ch | 80 BYTEs | default font 165 |
| 1ECh | 80 BYTEs | default user font |
| 23Ch | 80 BYTEs | default Pdir |
| 28Ch | 80 BYTEs | default sort |
| 2DCh | 10 BYTEs | default bill |
| 2E6h | 10 BYTEs | default route |
| 2F0h | 40 BYTEs | default cover sheet form |
| 318h | 34 BYTEs | valid dial characters |
| 33Ah | 6 BYTEs | local country code |
| 340h | 6 BYTEs | local city or area code |
| 346h | 6 BYTEs | future expansion |
| 34Ch | WORD | structure size |

**64-9 Format of STAT_IO_GET command buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | device number |
| 0Eh | WORD | current activity |
| 10h | WORD | number of rings (if ringing) |
| 12h | WORD | number of fascimiles transmitted |
| 14h | WORD | number of fascimiles received |
| 16h | WORD | status of last transmission |
| 18h | WORD | envelope number of last transmission |
| 1Ah | WORD | index of last destination in envelope |
| 1Ch | WORD | status of last reception |
| 1Eh | WORD | current page (if session in progress) |
| 20h | 80 BYTEs | current file |
| 70h | 104 BYTEs | remote number |
| D8h | 20 BYTEs | last name |
| ECh | 20 BYTEs | first name |
| 100h | 32 BYTEs | company name |
| 120h | 32 BYTEs | notes |
| 140h | WORD | current envelope ID (if sending) |
| 142h | WORD | total pages in transmission (if sending) |
| 144h | 6 BYTEs | future expansion |
| 14h | WORD | structure size |

**64-10 Format of STAT_FAXBIOS_GET command buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | status ID |
| 0Eh | WORD | currenty FaxBIOS function number |
| 10h | WORD | current Client ID being serviced |
| 12h | WORD | number of things to do |
| 14h | WORD | number of them done |
| 16h | WORD | number of pages to do (if any) |
| 18h | WORD | number of them done |
| 1Ah | WORD | number of files to do |

## INT 2Fh Function 80FBh

| | | |
|---|---|---|
| 1Ch | WORD | number of them done |
| 1Eh | 80 BYTEs | current File |
| 6Eh | WORD | 0 if all devices are idle |
| 70h | WORD | number of fascimiles transmitted |
| 72h | WORD | number of fascimiles received |
| 74h | WORD | status of last transmission in system |
| 76h | WORD | envelope ID of last transmission |
| 78h | WORD | index of last destination in envelope |
| 7Ah | WORD | status of last reception in system |
| 7Ch | DWORD | time of next transmission |
| 80h | 6 BYTEs | future expansion |
| 86h | WORD | structure size |

**64-11 Format of PDIR_OPEN command buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | 80 BYTEs | Phone Directory token |
| 5Ch | 80 BYTEs | sort order token |
| ACh | WORD | open Mode (0 = read, 1 = write) |
| AEh | WORD | Phone Directory handle |
| B0h | WORD | number of people |
| B2h | WORD | number of groups |
| B4h | WORD | bitmap of fields supported by partial read |
| B6h | 6 BYTEs | future expansion |
| BCh | WORD | structure size |

**64-12 Format of PDIR_CLOSE command buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | Phone Directory handle |
| 0Eh | 6 BYTEs | future expansion |
| 14h | WORD | structure size |

**64-13 Format of PDIR_READ_PERSON command buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | Phone Directory handle |
| 0Eh | WORD | retrieve by index |
| 10h | WORD | person ID or index |
| 12h | WORD | how many groups person is in |
| 14h | WORD | person ID |
| 16h | 20 BYTEs | last name |
| 2Ah | 20 BYTEs | first name |
| 3Eh | 32 BYTEs | company |
| 5Eh | 32 BYTEs | notes |
| 7Eh | 6 BYTEs | FAX country code |
| 84h | 6 BYTEs | FAX city/area code |
| 8Ah | 14 BYTEs | FAX local number |
| 98h | 14 BYTEs | FAX extension |
| A6h | 24 BYTEs | reserved |
| BEh | 6 BYTEs | voice country code |
| C4h | 6 BYTEs | voice city/area code |

| | | |
|---|---|---|
| CAh | 14 BYTEs | voice local number |
| D8h | 14 BYTEs | voice extension |
| E6h | 24 BYTEs | reserved |
| FEh | 10 BYTEs | outbound routing information |
| 108h | 10 BYTEs | billing information, credit card etc |
| 112h | DWORD | remote FAX capabilities |
| 116h | 21 BYTEs | T.30 poll code of FAX number |
| 12Bh | 15 BYTEs | reserved |
| 13Ah | 6 BYTEs | future expansion |
| 140h | WORD | structure size |

### 64-14 Format of SCHED_OPEN, SCHED_CANCEL command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | scheduler handle |
| 0Eh | 6 BYTEs | future expansion |
| 14h | WORD | structure size |

### 64-15 Format of SCHED_ADD_DEST command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | schedule handle |
| 0Eh | WORD | device number if manual send wanted |
| 10h | WORD | non-zero if poll desired |
| 12h | WORD | person ID |
| 14h | 20 BYTEs | last name |
| 28h | 20 BYTEs | first name |
| 3Ch | 32 BYTEs | company |
| 5Ch | 32 BYTEs | notes |
| 7Ch | 6 BYTEs | FAX country code |
| 82h | 6 BYTEs | FAX city/area code |
| 88h | 14 BYTEs | FAX local number |
| 96h | 14 BYTEs | FAX extension |
| A4h | 24 BYTEs | reserved |
| BCh | 6 BYTEs | voice country code |
| C2h | 6 BYTEs | voice city/area code |
| C8h | 14 BYTEs | voice local number |
| D6h | 14 BYTEs | voice extension |
| E4h | 24 BYTEs | reserved |
| FCh | 10 BYTEs | outbound routing information |
| 106h | 10 BYTEs | billing information, credit card etc |
| 110h | DWORD | remote FAX capabilities |
| 114h | 21 BYTEs | T.30 poll code of FAX number |
| 129h | 15 BYTEs | reserved |
| 138h | 6 BYTEs | future expansion |
| 13Eh | WORD | structure size |

### 64-16 Format of SCHED_ADD_FILE command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | schedule handle |
| 0Eh | WORD | file type |

| | | |
|---|---|---|
| | | 0000h unidentified, 0001h native file format,<br>0002h ASCII, 0003h FaxBIOS Tiff Class F |
| 10h | 80 BYTEs | file name |
| 60h | 80 BYTEs | font token |
| B0h | WORD | conversion options bitmap |
| B2h | WORD | resolution<br>0000h standard 98 lines per inch, 204 dpi<br>0001h fine 196 lines per inch, 204 dpi<br>0002h Group4 300 dpi<br>0003h Group4 400 dpi |
| B4h | WORD | page length<br>0000h 279mm (11 in), 0001h 297mm (11.69 in), 0002h 364mm (14.33 in)<br>0003h unlimited |
| B6h | WORD | page width<br>0000h 215mm (8.46 in), 0001h 255mm (10.04 in), 0002h 303mm (11.93 in),<br>0003h 151 mm (5.91 in), 0004h 107 mm (4.21 in) |
| B8h | WORD | binary file transfer specification<br>0000h only as FAX<br>0001h only as file (for non-faxable files)<br>0002h as file when possible else FAX |
| BAh | WORD | seam flag (nonzero for seam with next) |
| BCh | WORD | delete flag (nonzero to delete when done) |
| BEh | 6 BYTEs | future expansion |
| C4h | WORD | structure size |

**64-17 Format of SCHED_SET_PARAMS command buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | scheduler handle |
| 0Eh | DWORD | time to send |
| 10h | 10 BYTEs | priority token |
| 1Ch | 80 BYTEs | logo file token |
| 6Ch | 80 BYTEs | signature file token |
| BCh | 80 BYTEs | cover page token |
| 10Ch | 40 BYTEs | Subject text |
| 134h | 40 BYTEs | From text |
| 15Ch | WORD | user ID |
| 15Eh | 6 BYTEs | future expansion |
| 164h | WORD | structure size |

**64-18 Format of SCHED_CLOSE command buffer:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | scheduler handle |
| 0Eh | WORD | envelope ID generated |
| 10h | 6 BYTEs | future expansion |
| 16h | WORD | structure size |

### 64-19 Format of SLOG_OPEN, SLOG_CLOSE, RLOG_OPEN, RLOG_CLOSE command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | log handle |
| 0Eh | WORD | number of entries |
| 10h | 6 BYTEs | future expansion |
| 16h | WORD | structure size |

### 64-20 Format of GRAPH_GET_FILE_TYPE command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | 80 BYTEs | filename |
| 5Ch | WORD | file type (see Table 64-16) |
| 5Eh | WORD | bitmap of supported capabilities |
| 60h | 6 BYTEs | future expansion |
| 66h | WORD | structure size |

### 64-21 Format of GRAPH_CREATE_FILE command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | 80 BYTEs | filename |
| 5Ch | WORD | graph handle |
| 5Eh | 6 BYTEs | future expansion |
| 64h | WORD | structure size |

### 64-22 Format of GRAPH_CLOSE_FILE, GRAPH_END_PAGE command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | graph handle |
| 0Eh | 6 BYTEs | future expansion |
| 14h | WORD | structure size |

### 64-23 Format of GRAPH_CREATE_PAGE command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | graph handle |
| 0Eh | WORD | resolution |
| 10h | WORD | page width |
| 12h 6 | BYTEs | future expansion |
| 18h | WORD | structure size |

### 64-24 Format of GRAPH_WRITE_PAGE command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | graph handle |
| 0Eh | DWORD | pointer to storage for image |
| 12h | WORD | band height in lines |
| 14h | WORD | width of page image in bytes |
| 16h | WORD | facsimile page width constant |
| 18h | WORD | width of page image in bits |
| 1Ah | WORD | number of bytes actually processed |
| 1Ch | 6 BYTEs | future expansion |
| 22h | WORD | structure size |

## INT 2Fh Function 80FBh

### 64-25 Format of GRAPH_OPEN_FILE command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | 80 BYTEs | filename |
| 5Ch | WORD | file type (see Table 64-16) |
| 5Eh | WORD | graph handle |
| 60h | WORD | number of pages |
| 62h | 6 BYTEs | future expansion |
| 68h | WORD | structure size |

### 64-26 Format of GRAPH_GOTO_PAGE command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | graph handle |
| 0Eh | WORD | page number |
| 10h | WORD | vertical resolution |
| 12h | WORD | page width |
| 14h | DWORD | page length |
| 18h | 6 BYTEs | future expansion |
| 1Eh | WORD | structure size |

### 64-27 Format of GRAPH_READ_PAGE command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | graph handle |
| 0Eh | DWORD | pointer to storage for image |
| 12h | WORD | band height in lines |
| 14h | WORD | width of page image in bytes |
| 16h | WORD | facsimile page width constant |
| 18h | WORD | width of page image in bits |
| 1Ah | WORD | number of bytes actually processed |
| 1Ch | 6 BYTEs | future expansion |
| 22h | WORD | structure size |

### 64-28 Format of IOCTL_ANSWER_FAX command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | device number |
| 0Eh | 6 BYTEs | future expansion |
| 14h | WORD | structure size |

### 64-29 Format of IOCTL_DIAL command buffer:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | 12 BYTEs | common data (see Table 64-7) |
| 0Ch | WORD | device number |
| 0Eh | 6 BYTEs | country code |
| 14h | 6 BYTEs | city or area code |
| 1Ah | 14 BYTEs | local number |
| 28h | 14 BYTEs | extension |
| 36h | 14 BYTEs | reserved |
| 4Eh | 6 BYTEs | future expansion |
| 54h | WORD | structure size |

## MTEZ XpressFax

MTEZ Standard with ExpressFax, now owned by WordPerfect Corp., supports CAS, Class 1, and Class 2 fax modems (but in separate versions for each). The program offers background faxing together with an ability to capture print output from DOS programs and redirect it as a fax.

---

### INT 2Fh Function C0h

*MTEZ XpressFax Hardware TSR (CLASS2)*

***API***

**Registers at call:**
AH = C0h
AL = function code (01h to 15h)

**Return Registers:**
*unknown*

**Conflicts:** see Table 2-2
**See also:** Functions C000h and CB00h

---

### INT 2Fh Function C000h, Subfunction 444Bh

*MTEZ XpressFax Hardware TSR (CLASS2)*

***INSTALLATION CHECK***

**Registers at call:**
AX = C000h
BX = 444Bh (" DK")
CX = 4A4Eh (" JN")

**Return Registers:**
AL = status
    00h not installed, OK to install
    FFh installed
        BX = 646Bh (" dk")
        CX = 6A6Eh (" jn")

**Details:** This TSR will use any free multiplex number from C0h to FFh.
**Conflicts:** see Table 2-2
**See also:** Function C0h

---

### INT 2Fh Function CB00h, Subfunction 4D53h

*MTEZ XpressFax CASMGR*

***INSTALLATION CHECK***

**Registers at call:**
AX = CB00h
BX = 4D53h (" MS")
CX = 4949h (" II")

**Return Registers:**
AL = status
    00h not installed, OK to install
    01h not installed, not OK to install
    FFh installed
        BX = 6D73h (" ms")
        CX = 6969h (" ii")

**Details:** This function is equivalent to the standard CASMGR installation check, but uses the additional magic values to identify which CAS is installed.
**Conflicts:** none, but see Table 2-2
**See also:** Function C000h, Gammafax Function 2Ah (below), FaxBIOS Function 8000h (above), Communicating Applications Spec Functions CB0Eh and CB16h (Chapter 65)

## RainbowFAX

RainbowFax Pro is a product of Telimax Software, Vancouver, B.C., that is designed to send and receive color faxes between PCs using ordinary modems rather than special color FAX machines.

---

### INT 11h Function 77h

*RainbowFAX v1.3*

***SFENGINE API - OPERATIONAL CONTROL***

**Registers at call:**
AH = 77h
AL = subfunction
    01h request SFENGINE start

02h check if SFENGINE started

03h request SFENGINE stop

04h check if SFENGINE stopped

05h installation check

06h *uninstall*
BX:DX -> *return address for successful uninstall*

else

**Return Registers:**
ES:DX -> ASCIZ signature string " SFAX ENGINE V1.0"

AX = 0001h

AX = 0000h or 0001h

AX = 0001h

AX = 0000h or 0001h

AX = 0001h

(at caller's address)
AX = 0000h

AX = FFFFh

**See also:** Functions 78h, 79h, 7Ah, and 7Ch

---

### INT 11h Function 78h

*RainboxFAX v1.3*

**SFENGINE API - UNDETERMINED FUNCTIONS**

**Registers at call:**
AH = 78h
AL = subfunction (01h-03h)

**Return Registers:**
AX = status (0001h successful, FFFFh error)

**See also:** Functions 77h, 79h, 7Ah, and 7Ch

---

### INT 11h Function 79h

*RainboxFAX v1.3*

***SFENGINE API*** *- UNDETERMINED FUNCTIONS*

**Registers at call:**
AH = 79h
AL = subfunction

01h set *unknown* flag

02h clear *unknown* flag (see subfunc 01h)

03h set *unknown* flag (different from subfn 02h or 04h)

04h clear *unknown* flag (different from subfn 02h or 03h)

else

**Return Registers:**

AX = previous state (0000h already set, 0001h clear)

AX = 0001h

AX = 0001h

AX = 0001h

AX = FFFFh

**See also:** Functions 77h, 78h, 7Ah, and 7Ch

---

### INT 11h Function 7Ah

*RainboxFAX v1.3*

**SFENGINE API - UNDETERMINED FUNCTIONS**

**Registers at call:**
AH = 7Ah
AL = subfunction

01h set *unknown* flag

| | **Return Registers:** |
|---|---|
| 02h clear *unknown* flag | |
| 03h set *unknown* flag<br>(different from subfn 02h) | AX = previous state (0000h already set, 0001h clear) |
| | AX = 0001h |
| else | AX = 0001h |
| | AX = FFFFh |

**See also:** Functions 77h, 78h, 79h, and 7Ch

---

### INT 11h Function 7Ch
*RainboxFAX v1.3*

### *SFENGINE API - UNDETERMINED FUNCTIONS*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 7Ch | |
| AL = subfunction | |
| 01h set *unknown* flag<br>(see also subfn 02h) | AX = 0001h |
| | AX = 0001h |
| 02h clear *unknown* flag<br>(see also subfn 01h) | AX = FFFFh |
| else | |

**See also:** Functions 77h, 78h, 79h, and 7Ah

## ZyXEL ZFAX

ZFAX is the bundled FAX software which comes with the ZyXEL model fax modems. There are numerous potential conflicts with other programs using multiplex numbers in the range C0h through FFh; see Table 2-2 for details.

---

### INT 2Fh Function DA00h
*ZyXEL ZFAX*

### *INSTALLATION CHECK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DA00h | AL = 5Ah if installed<br>AH = state (00h enabled, 01h disabled) |

**See also:** Functions DA01h, DA02h, and DA03h; Comwave Function CBDCh (below)

---

### INT 2Fh Function DA01h
*ZyXEL ZFAX*

### *UNINSTALL*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DA01h | AL = status (00h success, 01h failure) |

**See also:** Function DA00h

---

### INT 2Fh Function DA02h
*ZyXEL ZFAX*

### *DISABLE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DA02h | AL = 00h |

**See also:** Function DA03h

---

### INT 2Fh Function DA03h
*ZyXEL ZFAX*

### *ENABLE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = DA03h | AL = 00h |

**See also:** Function DA02h

## Miscellaneous

---

**INT 2Fh Function 2Ah** *Gammafax DOS Dispatcher*

***INTERFACE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 2Ah | not known |

**Details:** Details were not available at the time of writing.

**See also:** Comwave Function CBDDh below, FaxBIOS Function 8000h (above), MTEZ Function C000h (above), BitFax INT 66h (above), Communicating Applications Spec Function CB00h (Chapter 65)

---

**INT 2Fh Function CAh** *FAXTSR*

***API***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = CAh<br>*others, if any, unknown* | *unknown* |

**Program:** FAXTSR is a resident fax send/receive module for FAXPLUS by Frans Veldman.

**Conflicts:** see Table 2-2

---

**INT 2Fh Function CBDCh** *Comwave Microfax Specification*

***GET STATUS ARRAY***

**Purpose:** Comwave manufacturers the Microfax line of fax cards which are a high-performance multi-card FAX solution for OEMs.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = CBDCh<br>BL = 56h<br>BH = line number (starts with 1)<br>CX:DX -> 32-byte status buffer or 80 character string | AX = number of cards installed, or FFFFh on error |

**Details:** The multiplex number defaults to AH=CBh but can be changed.

**Conflicts:** see Table 2-2

**See also:** Function CBDDh, ZFAX Function DA00h (above)

---

**INT 2Fh Function CBDDh** *Comwave Microfax Specification*

***COMMAND SUBMISSION***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = CBDDh<br>BL = 56h<br>BH = ?<br>CX:DX -> command to execute | AX = status<br>0000h success<br>FFFFh on error |

**See also:** Function CBDCh, Gammafax Function 2Ah, FaxBIOS Function 80FBh (above)

## CHAPTER 65

# *Communicating Applications Specification*

The DCA/Intel Communicating Applications Specification was developed as a joint effort by Digital Communications Associates, Inc. and Intel Corporation. It was originally intended to define a standard, high-level programming interface for data communications applications that would be independent of the hardware and software involved. It is by now the *de facto* standard for fax modem applications.

To be CAS-compliant, a system must provide a "Resident Manager" (either a TSR, a device driver, or a Windows DLL) that supports the functions listed in this chapter. Applications then use these functions to communication with the Resident Manager, which provides the actual hardware interfacing.

As with all software using multiplex numbers in the range C0h through FFh, there are multiple potential conflicts, which are listed in Table 2-2.

### INT 2Fh Function CB00h *CAS*

### *INSTALLATION CHECK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = CB00h | AL = status<br>00h not installed, OK to install<br>01h not installed, not OK to install<br>FFh installed |

**Details:** AH = CBh is the default multiplex identifier, but may be reconfigured.

**See also:** Function CB0Eh, Gammafax Function 2Ah (Chapter 64), FaxBIOS Function 8000h (Chapter 64), Comwave Function CBDCh (Chapter 64)

### INT 2Fh Function CB01h *CAS*

### *SUBMIT A TASK*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = CB01h | AX >= 0: event handle |
| DS:DX -> ASCIZ name of task control file | AX < 0: error code (see Table 65-1) |

**Details:** Files needed for an event must be kept until the task is complete or an error occurs.

**See also:** Functions CB0Bh and CB15h

**65-1 Values for error codes (AH = class, AL = subcode, value passed back is 2's complement):**

| | |
|---|---|
| Class 00h | --- FAX warnings (see Table 65-2) |
| Class 01h | --- DOS warnings (data was sent) |
| Subcode 01h | invalid function |
| 05h | access denied |
| 06h | invalid handle |
| others | see Table 24-2 |

| | |
|---|---|
| Class 02h | --- fatal errors (data not sent) (see Table 65-3) |
| Class 03h | --- fatal DOS errors |
| Subcode 02h | file not found |
| 03h | path not found |
| others | see Table 24-2 |
| Class 04h | --- FAX errors (see Table 65-4) |
| Class 05h | --- application-specific (v1.2+) |
| | ---Intel FAXPOP.EXE |
| Subcode 00h | tried to send while in graphics mode |
| 01h | insufficient disk space |
| 02h | internal buffer overflow |
| Class 06h | --- CAS implementation-specific (v1.2+) |

**65-2 FAX warning subcodes:**

| | |
|---|---|
| 00h | no error |
| 02h | bad scanline count |
| 03h | page sent with errors, could not retransmit |
| 04h | received data lost |
| 05h | invalid or missing logo file |
| 06h | filename does not match nonstandard format (NSF) header |
| 07h | file size does not match NSF header |

**65-3 Fatal error subcodes:**

| | |
|---|---|
| 00h | multiplex handler failed |
| 01h | unknown command |
| 02h | bad event handle |
| 03h | FIND NEXT attempted before FIND FIRST |
| 04h | no more events |
| 07h | invalid queue type |
| 08h | bad control file |
| 09h | communication board busy |
| 0Ah | invalid command parameter |
| 0Bh | can't uninstall resident code |
| 0Ch | file exists |
| 80h | unknown task type |
| 81h | bad phone number |
| 82h | bad .PCX file header |
| 83h | unexpected EOF |
| 84h | unexpected disconnect |
| 85h | too many dialing retries |
| 86h | no file specified for send |
| 87h | communication board timeout |
| 88h | received too many pages (>1023) of data |
| 89h | manual connect initiated too long ago |
| 8Ah | hardware command set error |
| 8Bh | bad NonStandard Format (NSF) header file |

**65-4 FAX error subcodes:**

| | |
|---|---|
| 01h | remote unit not Group 3 compatible |
| 02h | remote unit did not send capabilities |
| 03h | other FAX machine incompatible |
| 04h | other FAX incapable of file transfers |
| 05h | exceeded retrain or FAX resend limit |

## INT 2Fh Function CB01h

| | |
|---|---|
| 06h | line noise or failure to agree on bit rate |
| 07h | remote disconnected after receiving data |
| 08h | no response from remote after sending data |
| 09h | remote's capabilities incompatible |
| 0Ah | no dial tone (v1.2+) |
| 0Bh | invalid response from remote unit after sending data |
| 0Dh | phone line dead or remote unit disconnected |
| 0Eh | timeout while waiting for secondary dial tone (v1.2+) |
| 11h | invalid command from remote after receiving data |
| 15h | tried to receive from incompatible hardware |
| 5Ch | received data overflowed input buffer |
| 5Dh | remote unexpectedly stopped sending data |
| 5Eh | other FAX machine jammed (no data sent) |
| 5Fh | remote took too long to send fax scan line |
| 63h | can't get through to remote unit |
| 64h | user canceled event |

---

## INT 2Fh Function CB02h *CAS*

### *ABORT CURRENT EVENT*

**Registers at call:**
AX = CB02h

**Return Registers:**
AX >= 0 event handle of aborted event
AX < 0 error code (see Table 65-1)

**Details:** Termination could take up to 30 seconds.
**See also:** Functions CB08h and CB10h

---

## INT 2Fh Function CB05h *CAS*

### *FIND FIRST QUEUE ENTRY*

**Registers at call:**
AX = CB05h
CX = status of events to find (see Table 65-5)
DH = direction
 00h chronological order, earliest to latest
 01h reverse chronological order, latest to earliest
DL = queue to search (00h-02h) (see Table 65-6)

**Return Registers:**
AX = 0000h successful
 BX = event handle for found event
AX < 0 error code (see Table 65-1)

**See also:** Functions CB06h and CB07h
.comment snake into two columns

**65-5 Events to be found:**

| | |
|---|---|
| 0000h | successful completion |
| 0001h | waiting to be processed |
| 0002h | number has been dialed |
| 0003h | connection established, sending |
| 0004h | connection established, receiving |
| 0005h | event aborted |
| FFFFh | find any event, regardless of status |
| other | negative values, match error code |

**65-6 Queue identifiers:**

| | |
|---|---|
| 00h | task queue |
| 01h | receive queue |
| 02h | log queue |
| 03h | group file in task queue (v1.2+) |
| 04h | group file in log queue (v1.2+) |

### INT 2Fh Function CB06h

*CAS*

***FIND NEXT QUEUE ENTRY***

**Registers at call:**
AX = CB06h
DL = queue to search (00h-02h) (see Table 65-6)

**Return Registers:**
AX = 0000h successful
    BX = event handle for found event
AX < 0 error code (see Table 65-1)

**Details:** The direction of search is the same as for the preceding FIND FIRST call.

**See also:** Function CB05h

### INT 2Fh Function CB07h

*CAS*

***OPEN FILE***

**Registers at call:**
AX = CB07h
BX = event handle from find (Functions CB05h, CB06h) or submit task (FunctionCB01h)
CX = receive file number (ignored for task queue and log queue)
    0000h open receive control file
    N open Nth received data file
DL = queue (see Function CB06h)

**Return Registers:**
AX = 0000h successful
    BX = DOS file handle for requested file
AX < 0 error code (see Table 65-1)

**Details:** The returned file handle has been opened in read-only mode and should be closed with INT 21h Function 3Eh after use.

**See also:** Functions CB01h, CB05h, and CB14h

### INT 2Fh Function CB08h

*CAS*

***DELETE FILE***

**Registers at call:**
AX = CB08h
BX = event handle
CX = receive file number
    0000h delete *all* received files and receive control file
    N delete Nth received file
DL = queue
    00h delete control file in task queue and corresponding group file if it exists
    01h delete file in receive queue, as given by CX
    02h delete control file in log queue (individual deletions not recommended, to maintain integrity of log) and corresponding group file if it exists

**Return Registers:**
AX = 0000h successful
AX < 0 error code (see Table 65-1)

**See also:** Functions CB02h and CB09h

### INT 2Fh Function CB09h

*CAS*

***DELETE ALL FILES IN QUEUE***

**Registers at call:**
AX = CB09h
DL = queue
    00h task queue (control and group files)
    01h receive queue
    02h log queue (control and group files)

**Return Registers:**
AX = 0000h successful
AX < 0 error code (see Table 65-1)

**See also:** Function CB08h

### INT 2Fh Function CB0Ah

*CAS*

***GET EVENT DATE***

**Registers at call:**
AX = CB0Ah
BX = event handle
DL = queue (00h-02h) (see Table 65-6)

**Return Registers:**
AX = 0000h successful
    CX = year
    DH = month

DL = day
AX < 0 error code (see Table 65-1)

**See also:** Functions CB0Bh and CB0Ch

---

### INT 2Fh Function CB0Bh
CAS

**SET TASK DATE**

**Registers at call:**
AX = CB0Bh
BX = event handle (task event only)
CX = year
DH = month
DL = day

**Return Registers:**
AX = 0000h successful
AX < 0 error code (see Table 65-1)

**Details:** Setting a task's date and time to before the current date and time causes it to execute immediately.

**See also:** Functions CB01h, CB0Ah, and CB0Dh

---

### INT 2Fh Function CB0Ch
CAS

**GET EVENT TIME**

**Registers at call:**
AX = CB0Ch
BX = event handle
DL = queue (00h-02h) (see Table 65-6)

**Return Registers:**
AX = 0000h successful
CH = hour
CL = minute
DH = second
DL = 00h
AX < 0 error code (see Table 65-1)

**See also:** Functions CB0Ah and CB0Dh

---

### INT 2Fh Function CB0Dh
CAS

**SET TASK TIME**

**Registers at call:**
AX = CB0Dh
BX = event handle (task events only)
CH = hour
CL = minute
DH = second
DL unused

**Return Registers:**
AX = 0000h successful
AX < 0 error code (see Table 65-1)

**Details:** Setting a task's date and time to a value before the current date and time causes it to execute immediately.

**See also:** Functions CB0Bh, CB0Ch, and CB10h

---

### INT 2Fh Function CB0Eh
CAS

**GET EXTERNAL DATA BLOCK**

**Registers at call:**
AX = CB0Eh
DS:DX -> 256-byte buffer

**Return Registers:**
AX = 0000h successful
buffer filled
AX < 0 error code (see Table 65-1)

**65-7 Format of external data block:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | CAS major version |
| 01h | BYTE | CAS minor version |
| 02h | 68 BYTEs | ASCIZ path to directory containing CAS software, ends in slash |
| 46h | 13 BYTEs | ASCIZ name of current phonebook (in CAS directory) |
| 53h | 13 BYTEs | ASCIZ name of current logo file (in CAS directory) |
| 60h | 32 BYTEs | ASCIZ default sender name |

| | | |
|---|---|---|
| 80h | 21 BYTEs | ASCIZ CCITT identification of fax device |
| 95h | 107 BYTEs | reserved |

---

### INT 2Fh Function CB0Fh

CAS

### *GET/SET AUTORECEIVE*

**Registers at call:**
AX = CB0Fh
DL = subfunction
  00h get current autoreceive state
  01h set autoreceive state
    DH = number of rings before answer, 00h = never

**Return Registers:**
AX = status
  0000h autoreceive disabled
  N number of rings before answer
AX < 0 error code (see Table 65-1)

---

### INT 2Fh Function CB10h

CAS

### *GET CURRENT EVENT STATUS*

**Registers at call:**
AX = CB10h
DS:DX -> 512-byte buffer

**Return Registers:**
AX = 0000h successful
  BX = event handle of current event or negative error code if no current event
  buffer filled
AX < 0 error code (see Table 65-1)

**See also:** Functions CB02h and CB0Dh

**65-8 Format of status area:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | event type |
| | | 00h send, 01h receive, 02h polled send, 03h polled receive |
| | | 04h to 7Fh reserved |
| | | FFh serious hardware error |
| 01h | BYTE | transfer type |
| | | 00h 200x200 dpi, FAX mode |
| | | 01h 100x200 dpi, FAX mode |
| | | 02h file transfer mode |
| | | 03h to 7Fh reserved |
| 02h | WORD | event status (see below) |
| 04h | WORD | event time (packed DOS time format, see INT 21h Function 5700h) |
| 06h | WORD | event date (packed DOS date format, see INT 21h Function 5700h) |
| 08h | WORD | number of files to transfer, max 7FFFh |
| 0Ah | WORD | offset of file transfer record |
| 0Ch | 47 BYTEs | ASCIZ phone number to call |
| 3Bh | 64 BYTEs | ASCIZ application-specific tag string |
| 7Bh | BYTE | reserved (00h) |
| 7Ch | BYTE | connect time, seconds |
| 7Dh | BYTE | connect time, minutes |
| 7Eh | BYTE | connect time, hours |
| 7Fh | DWORD | total number of pages in all files |
| 83h | DWORD | pages already transmitted |
| 87h | WORD | number of files already transmitted |
| 89h | BYTE | cover page flag |
| | | 00h don't transmit cover page |
| | | 01h transmit cover page |
| | | 02h to 7Fh reserved |
| 8Ah | WORD | total number of transmission errors |

| | | |
|---|---|---|
| 8Ch | 78 BYTEs | reserved (zeros) |
| DAh | 21 BYTEs | ASCIZ remote FAX's CCITT identification |
| EFH | 32 BYTEs | ASCIZ destination name |
| 10Fh | 32 BYTEs | ASCIZ sender name |
| 12Fh | 80 BYTEs | filename of PCX logo file (max 1780x800 pixels) |
| 17Fh | 128 BYTEs | file transfer record for current event (see below) |

### 65-9 Values for event status:

| | |
|---|---|
| 0000h | completed successfully |
| 0001h | waiting |
| 0002h | number dialed |
| 0003h | connected, sending |
| 0004h | connected, receiving |
| 0005h | aborted |
| 0006h to 007Fh | reserved |
| 0080h to 7FFFh | application-specific events |
| 8000h to FFFFh | error codes |

### 65-10 Format of file transfer record:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | file type (ignored unless FAX)<br>00h ASCII, 01h PCX, 02h DCX, 03h to 7Fh reserved |
| 01h | BYTE | text size for ASCII FAX file<br>00h = 80 columns by 66 lines (11 inches)<br>01h = 132 columns by 88 lines (11 inches)<br>02h to 7Fh reserved |
| 02h | BYTE | status of file (see below) |
| 03h | DWORD | bytes already transmitted |
| 07h | DWORD | file size in bytes |
| 0Bh | WORD | pages alread transmitted |
| 0Dh | WORD | number of pages in file |
| 0Fh | 80 BYTEs | ASCIZ filename |
| 5Fh | BYTE | 1/8 inch page length<br>if page length below set to 01h through 7Fh, this value specifies additional 1/8 inch increments to page length |
| 60h | BYTE | page length<br>00h = 11 inches<br>01h to 7Fh = page length is this number of inches plus value of 1/8 inch field above<br>80h to FEh reserved<br>FFh = ASCII pages ending with formfeed |
| 61h | 31 BYTEs | reserved (zeros) |

### 65-11 Values for file status:

| | |
|---|---|
| 00h | untouched |
| 01h | opened |
| 02h | moved |
| 03h | deleted |
| 04h | not yet received |
| 05h | to 7Fh reserved |

## INT 2Fh Function CB11h

*CAS*

### *GET QUEUE STATUS*

**Registers at call:**
AX = CB11h
DL = queue for which to get status
00h task queue
01h receive queue
02h log queue

**Return Registers:**
AX >= 0 total number of changes made to queue, modulo 32768
BX = number of control files currently in queue
CX = number of received files (zero for task and log queues)
AX < 0 error code (see Table 65-1)

**See also:** Function CB11h Subfunctions 03h and 04h, Function CB12h

## INT 2Fh Function CB11h, Subfunction 03h

*CAS v1.2+*

### *GET NUMBER OF SEND EVENTS*

**Registers at call:**
AX = CB11h
DL = 03h

**Return Registers:**
AX = number of successful sends since resident manager started
BX = number of unsuccessful sends, including warnings

**See also:** Function CB11h Subfunction 04h

## INT 2Fh Function CB11h, Subfunction 04h

*CAS v1.2+*

### *GET NUMBER OF RECEIVE EVENTS*

**Registers at call:**
AX = CB11h
DL = 04h

**Return Registers:**
AX = number of received file events since resident manager started
BX = number of received FAX events

**See also:** Function CB11h Subfunction 03h

## INT 2Fh Function CB12h

*CAS*

### *GET HARDWARE STATUS*

**Registers at call:**
AX = CB12h
DS:DX -> 128-byte status buffer (see Tables 65-12 and 65-13)

**Return Registers:**
AX = 0000h successful
buffer filled with hardware-dependent status information
AX < 0 error code (see Table 65-1)

**See also:** Functions CB10h and CB11h

**65-12 Format of status buffer for Intel Connection CoProcessor:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | Connection CoProcessor connection status flags (see below) |
| 01h | BYTE | number of kilobytes of free buffer space |
| 02h | BYTE | page buffer status<br>bit 7: Connection CoProcessor has documents to send<br>bits 6-0: number of pages in buffer |
| 03h | BYTE | number of retries left for dialing number |
| 04h | BYTE | page number to retransmit |
| 05h | BYTE | communications status (see below) |
| 06h | BYTE | baud rate<br>bit 7: reserved<br>bits 6-4: baud rate<br>000 = 300 baud (V.21 SDLC or HDLC mode)<br>100 = 2400 baud (V.27 ter)<br>101 = 4800 baud (V.27 ter)<br>110 = 7200 baud (V.29)<br>111 = 9600 baud (V.29)<br>bits 3-0: reserved, should be 0110 |

| | | |
|---|---|---|
| 07h | 3 BYTEs | reserved |
| 0Ah | BYTE | Connection CoProcessor hardware status (see below) |
| 0Bh | BYTE | Connection CoProcessor switch states (see below) |
| 0Ch | BYTE | communications flags (see below) |
| 0Dh | BYTE | reserved |
| 0Eh | WORD | error count (only valid while busy, reset when idle) |
| 10h | DWORD | size of nonstandard format (NSF) file in bytes |
| 14h | BYTE | 'A' if Connection CoProcessor board present |
| 15h | 9 BYTEs | reserved |
| 1Eh | 21 BYTEs | ASCIZ CCITT identification |
| 33h | 77 BYTEs | reserved |

**Note:** The Intel Connection CoProcessor and SatisFAXtion may be distinguished by examining the byte at offset 14h.

### 65-13 Format of status buffer for Intel SatisFAXtion board:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | SatisFAXtion connection status flags (see below) |
| 01h | BYTE | SatisFAXtion board state (see below) |
| 02h | BYTE | number of KB free in buffer |
| 03h | BYTE | number of pages or files in buffer |
| 04h | BYTE | number of redials remaining on current number |
| 05h | BYTE | FAX page number to retransmit |
| 06h | BYTE | current page/file in block transfer |
| 07h | BYTE | number of rings received (only if auto-answer enabled) |
| 08h | WORD | error count |
| 0Ah | DWORD | length of file being transferred |
| 0Eh | 6 BYTEs | reserved |
| 14h | BYTE | 'B' if SatisFAXtion board present |
| 15h | 13 BYTEs | ASCIZ transfer agent name |
| 22h | 5 BYTEs | ASCIZ transfer agent version number |
| 27h | 13 BYTEs | ASCIZ resident loader name |
| 34h | 5 BYTEs | ASCIZ resident loader version number |
| 39h | 21 BYTEs | ASCIZ remote CSID |
| 4Eh | 13 BYTEs | ASCIZ resident manager name |
| 5Bh | 5 BYTEs | ASCIZ resident manager version number |
| 60h | 32 BYTEs | reserved |

### 65-14 Bitfields for Connection CoProcessor connection status flags:

| | |
|---|---|
| bit 7 | hardware busy sending or receiving |
| bit 6 | last page of data |
| bit 5 | no data on current page |
| bit 4 | retransmit request for current page being transmitted |
| bit 3 | NSF (nonstandard file) mode active |
| bits 2-0 | reserved |

### 65-15 Bitfields for communications status:

| | |
|---|---|
| bit 7 | originating call |
| bit 6 | FAX message to be sent |
| bit 5 | on line |
| bit 4 | ring detected and receive enabled |
| bit 3 | buffer dumped on receive |
| bits 2-0 | hardware sequence state |
| | 000 idle |

| | |
|---|---|
| 001 | dial |
| 010 | answer |
| 011 | transmit |
| 100 | receive |
| 101 | pre-message |
| 110 | post-message |
| 111 | disconnect |

### 65-16 Bitfields for Connection CoProcessor hardware status:

| | |
|---|---|
| bit 7 | modem option installed |
| bit 6 | Connection CoProcessor has control of DAA (not latched) |
| bit 5 | on line (not latched) |
| bit 4 | ring detected (not latched) |
| bit 3 | data in command buffer (not latched) |
| bit 2 | set if using DMA channel 1, clear if using DMA channel 3 |
| bit 1 | line length compensation bit 1 set (not latched) |
| bit 0 | line length compensation bit 0 set (not latched) |

### 65-17 Bitfields for Connection CoProcessor switch states:

bit 7: reserved
bit 6: unused
bit 5: spare switch open
bit 4: FAX ADR1 switch open
bit 3: FAX ADR0 switch open
bit 2: alternate interrupt switch open
bit 1: COM SEL 1 switch open
bit 0: COM SEL 0 switch open

**Note:** valid combinations of bits 0-2 are

| | |
|---|---|
| 000 | COM2 IRQ3 IObase 2F8h |
| 001 | COM1 IRQ4 IObase 3F8h |
| 010 | COM4 IRQ3 IObase 2E8h |
| 011 | COM3 IRQ4 IObase 3E8h |
| 110 | COM4 IRQ2 IObase 2E8h |
| 111 | COM3 IRQ5 IObase 3E8h |

### 65-18 Bitfields for communications flags:

| | |
|---|---|
| bit 7 | reserved |
| bit 6 | auxiliary relay forced ON |
| bit 5 | modem select relay forced ON |
| bit 4 | offhook relay forced ON |
| bit 3 | 9600 bps enabled |
| bit 2 | 7200 bps enabled |
| bit 1 | 4800 bps enabled |
| bit 0 | 2400 bps enabled |

### 65-19 Bitfields for SatisFAXtion connection status flags:

| | |
|---|---|
| bit 7 | busy in T.30 CCITT fax protocol |
| bit 6 | data on current page/file (only used for block xfers) |
| bit 5 | retransmission of last page requested |
| bit 4 | in file transfer mode |
| bit 3 | data in buffer |
| bit 2 | data buffer dumped on receive |
| bit 1 | 200x100 dpi resolution instead of 200x200 dpi |
| bit 0 | data modem in use, FAX image modem not available |

## INT 2Fh Function CB12h

### 65-20 Bitfields for SatisFAXtion board state:

| | |
|---|---|
| bit 7 | reserved |
| bit 6 | handset jack active, data and FAX modems not available |
| bits 5-3 | current bit rate<br>000 300 bps (V.21 HDLC)<br>100 2400 bps (V.27 ter)<br>101 4800 bps (V.27 ter)<br>110 7200 bps (V.29)<br>111 9600 bps (V.29) |
| bits 2-0 | T.30 CCITT protocol state<br>000 idle, 001 dialing, 010 answering, 011 transmitting,<br>100 receiving, 101 pre-message, 110 post-message, 111 disconnect |

### INT 2Fh Function CB13h, Subfunction 00h

*CAS*

#### *GET DIAGNOSTICS RESULTS*

| Registers at call: | Return Registers: |
|---|---|
| AX = CB13h | AX = status |
| DL = 00h | 0040h in progress<br>>= 0 passed<br>< 0 hardware-dependent failure code (see Tables 65-21 and 65-22) |

### 65-21 Intel Connection CoProcessor failure codes:

| | |
|---|---|
| bit 3 | 9600 bps FAX modem module failed |
| bit 2 | SDLC chip failed |
| bit 1 | RAM failed |
| bit 0 | ROM checksum failed |

### 65-22 Intel SatisFAXtion failure codes:

| | |
|---|---|
| bit 1 | 2400 bps data modem failed |
| bit 0 | 9600 bps FAX modem failed |

**See also:** Function CB13h Subfunction 01h

### INT 2Fh Function CB13h, Subfunction 01h

*CAS*

#### *START DIAGNOSTICS*

| Registers at call: | Return Registers: |
|---|---|
| AX = CB13h | AX = 0000h successfully started |
| DL = 01h | AX < 0 error code (see Table 65-1) |

**See also:** Function CB13h Subfunction 00h

### INT 2Fh Function CB14h

*CAS*

#### *MOVE RECEIVED FILE*

| Registers at call: | Return Registers: |
|---|---|
| AX = CB14h | AX = 0000h successful |
| BX = event handle | AX < 0 error code (see Table 65-1) |
| CX = receive file number<br>0001h first received file<br>N Nth received file | |
| DS:DX -> ASCIZ string specifying new name for file (must not exist) | |

### INT 2Fh Function CB15h

*CAS*

#### *SUBMIT FILE TO SEND*

| Registers at call: | Return Registers: |
|---|---|
| AX = CB15h | AX >= 0 event handle |
| DS:DX -> variable-length data area (see Table 65-23) | AX < 0 error code (see Table 65-1) |

**See also:** Function CB01h

**65-23 Format of data area:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | transfer type<br>00h = 200x200 dpi, fax mode<br>01h = 100x200 dpi, fax mode<br>02h = file transfer mode<br>03h to 7Fh reserved |
| 01h | BYTE | text size<br>00h = 80 columns<br>01h = 132 columns<br>02h to 7Fh reserved |
| 02h | WORD | time to send (DOS packed time format, see Table 24-!I) |
| 04h | WORD | date to send (DOS packed date format, see Table 24-!J) |
| 06h | 32 BYTEs | ASCIZ destination name |
| 26h | 80 BYTEs | ASCIZ name of file to send |
| 76h | 47 BYTEs | ASCIZ phone number to dial |
| A5h | 64 BYTEs | ASCIZ application-specific tag string |
| E5h | BYTE | reserved (00h) |
| E6h | BYTE | cover page (00h don't send, 01h send cover, 02h to 7Fh reserved) |
| E7h | 23 BYTEs | reserved (zeros) |
| FEh | variable | ASCIZ string containing text of cover page (if cover page flag set to 01h) |

## INT 2Fh Function CB16h, Subfunction 1234h

*CAS v1.2+*

### *UNLOAD RESIDENT MANAGER*

| Registers at call: | Return Registers: |
|---|---|
| AX = CB16h | AX = 0000h successful |
| BX = 1234h | AX < 0 error code (see Table 65-1) |
| CX = 5678h | |
| DX = 9ABCh | |

**Details:** The MTEZ XpressFax CASMGR ignores the values in BX,CX, and DX.

**See also:** Function CB00h, MTEZ Function C000h (Chapter 64)

## INT 2Fh Function CB17h

*CAS v1.2+*

### *SET COVER PAGE STATUS*

| Registers at call: | Return Registers: |
|---|---|
| AX = CB17h | AX = 0000h successful |
| BX = event handle | AX < 0 error code (see Table 65-1) |
| CL = cover page status (00h not read, 01h read by user) | |

## INT 2Fh Function CB80h to CB82h

*Intel SatisFAXtion CASMGR*

### *UNDETERMINED FUNCTION*

| Registers at call: | Return Registers: |
|---|---|
| AX = CB80h to CB82h | *unknown* |
| *others, if any, unknown* | |

## CHAPTER 66

# *Miscellaneous APIs*

This final chapter describes those functions which do not neatly fit into any of the other chapters. It begins with a section for those programs providing only a single call, and then continues with individual sections (in alphabetical order by product name) for various other programs which did not fit elsewhere.

## Installation Checks only

Many products hook an interrupt solely to provide an installation check, and otherwise either provide no calls at all or provide only standard calls which are listed in other chapters. These products are covered in this section.

---

### INT 12h

*PARKER v2.0+*

***INSTALLATION CHECK***

**Program:** PARKER is an optionally-resident hard disk parking program by Johan Zwiekhorst

| Registers at call: | Return Registers: |
|---|---|
| CX = 1807h | AX = kilobytes of contiguous memory starting at absolute address 00000h<br>CX = 1961h if installed |

**Conflicts:** BIOS (Chapter 4), Back&Forth (Chapter 49), Borland C++ IDE (Chapter 54)

---

### INT 16h Function 5500h

*Microsoft Word internal Callout*

***MICROSOFT WORD COOPERATION WITH TSR***

| Registers at call: | Return Registers: |
|---|---|
| AX = 5500h | AX = 4D53h ('MS') if keyboard TSR present |

**Details:** During startup, Microsoft Word tries to communicate with any TSRs that are present through this call. If the return is not 4D53h, Word installs its own INT 09h and INT 16h handlers; otherwise it assumes that the TSR will handle the keyboard.

**Conflicts:** Swap Utilities (Chapter 51)

**See also:** Word Perfect INT 1Ah Function 3601h

---

### INT 16h Function 6C63h

*TMED v1.6a (undocumented)*

***INSTALLATION CHECK***

**Program:** TMED is a freeware resident memory editor by Liang Chen.

| Registers at call: | Return Registers: |
|---|---|
| AX = 6C63h ('lc') | AX = 4C43h ('LC') if installed |

---

### INT 16h Function 7761h

*WATCH.COM v2.x-v3.0*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = 7761h ('wa') | AX = 5741h ('WA') if installed |

**Details:** WATCH.COM is part of the "TSR" package by Kim Kokkonen.

**Conflicts:** pcANYWHERE (Chapter 58)
**See also:** INT 21h Function 7761h

---

### INT 16h Function 7788h
*PC Magazine PUSHDIR.COM*

***INSTALLATION CHECK***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 7788h | AX = 7789h if installed and signature correct |
| BX = 7789h | BX = 7788h |
| DS:SI -> signature "PUSHDIR VERSION 1.0" | SI destroyed |

**Conflicts:** pcANYWHERE (Chapter 58)

---

### INT 16h Function 80h
*MAKEY.COM*

***INSTALLATION CHECK***

**Program:** MAKEY is a utility by System Enhancement Associates

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 80h | AX = 1954h if installed |

---

### INT 16h Function AABBh
*JORJ v4.3*
*(undocumented)*

***INSTALLATION CHECK***

**Program:** JORJ is a shareware dictionary with phonetic lookup by Jorj Software Co.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = AABBh | AX = EEFFh if installed |
| BX = EEFFh | BX = AABBh if installed |
| | CL = hotkey name (default 6Ah 'j' for Alt-J) |

**Conflicts:** PTxxx.COM (Chapter 54)

---

### INT 16h Function EFh
*CALCULATOR*

***INSTALLATION CHECK***

**Program:** CALCULATOR is a shareware popup calculator by Andrzej Brzezinski and Marek Kosznik.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = EFh | AX = 0088h if installed |

---

### INT 1Ah Function 3601h
*Word Perfect v5.0 Third Party Interface Callout*

***INSTALLATION CHECK***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 3601h | DS:SI = routine to monitor keyboard input, immediately preceded by the ASCIZ string "WPCORP\0" |

**Details:** WordPerfect 5.0 will call this interrupt at start up to determine if a third party product wants to interface with it. The third party product must intercept this interrupt and return the address of a keyboard monitor routine.

Before checking for keyboard input, and after every key entered by the user, Word Perfect will call the routine whose address was provided in DS:SI with the following parameters:

Entry: AX = key code or 0
BX = WordPerfect state flag
Exit: AX = 0 or key code
BX = 0 or segment address of buffer with key codes.

See the "WordPerfect 5.0 Developer's Toolkit" for further information.

**See also:** Microsoft Word INT 16h Function 5500h

---

### INT 1Ah Function CCCCh
*DATEFIX*
*(undocumented)*

***INSTALLATION CHECK***

**Program:** DATEFIX is a public-domain TSR to correct the date on AT&T 6300 machines, where the real-time clock's calendar wraps after 1991.

**Registers at call:**
AX = CCCCh
BX = CCCCh
CX = 0000h

**Return Registers:**
CX = CCCCh if installed
    ES:BX -> original interrupt handler

**See also:** AT&T 6300 Functions FEh and FFh (Chapter 7)

---

### INT 21h Function 2Bh, Subfunction 454Ch

*ELRES v1.1*

### *INSTALLATION CHECK*

**Program:** ELRES is an MS-DOS return code (errorlevel) recorder by David H. Bennett which stores recent errorlevel values, allows them to be retrieved for use in batch files, and can place them in an environment variable.

**Registers at call:**
AH = 2Bh
CX = 454Ch ('EL')
DX = 5253h ('RS')

**Return Registers:**
ES:BX -> ELRES history structure (see below)
DX = DABEh (signature, DAve BEnnett)

**See also:** ELRES Function 4Bh below, MS-DOS Function 4Dh (Chapter 24)

**66-1 Format of ELRES history structure:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | number of return codes which can be stored by following buffer |
| 02h | WORD | current position in buffer (treated as a ring) |
| 04h | N BYTEs | ELRES buffer |

---

### INT 21h Function 2B01h, Subfunction 5441h

*TAME v2.10+*

### *INSTALLATION CHECK*

**Program:** TAME is a formerly-shareware (commercial since version 3.0) program by David G. Thomas which gives up CPU time to other partitions under a multitasker when the current partition's program incessantly polls the keyboard or system time.

**Registers at call:**
AX = 2B01h
CX = 5441h ('TA')
DX = 4D45h ('ME')
---v2.60---
BH = *unknown*
    00h skip *unknown item*, else do

**Return Registers:**
AL = 02h if installed
ES:DX -> data area in TAME-RES (see below)

**66-2 Format of TAME 2.60 data area:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | data structure minor version number (02h in TAME 2.60) |
| 01h | BYTE | data structure major version number (0Bh in TAME 2.60) |
| 02h | DWORD | number of task switches |
| 06h | DWORD | number of keyboard polls |
| 0Ah | DWORD | number of time polls |
| 0Eh | DWORD | number of times DESQview told program runs only in foreground |
| 12h | DWORD | time of last /CLEAR or TAME-RES load |
| 16h | DWORD | time yielded |
| 1Ah | DWORD | time spent polling |
| 1Eh | DWORD | time spent waiting on key input with INT 16h Function 01h,11h |
| 22h | 4 BYTEs | *unknown* |
| 26h | 7 DWORDs | original INT 10h,INT 14h,INT 15h,INT 16h,INT 17h,INT 21h,INT 28h |
| 42h | 7 WORDs | offsets of TAME INT 10h,14h,15h,16h,17h,21h, and 28h handlers |
| 50h | WORD | X in /max:X,Y or /freq:X,Y |
| 52h | WORD | Y in /max:X,Y or /freq:X,Y |

| | | |
|---|---|---|
| 54h | WORD | number of polls remaining before next task switch |
| 56h | WORD | /KEYIDLE value |
| 58h | 4 BYTEs | *unknown* |
| 5Ch | WORD | X in /boost:X,Y |
| 5Eh | WORD | Y in /boost:X,Y |
| 60h | WORD | /FG: value |
| 62h | WORD | task switches remaining until next FGONLY DESQview API call |
| 64h | WORD | *multitasker version* |
| 66h | WORD | virtual screen segment |
| 68h | BYTE | interrupts already grabbed by TAME (see below) |
| 69h | BYTE | flags for interrupts which may be acted on (same bits as above) |
| 6Ah | BYTE | window or task number for this task |
| 6Bh | BYTE | multitasker type (see below) |
| 6Ch | BYTE | type of task switching selected (bit flags) (see below) |
| 6Dh | BYTE | watch_DOS |
| 6Eh | BYTE | bit flags<br>bit 0: TAME enabled<br>bit 1: /FREQ instead of /MAX (counts in 50h and 52h per tick)<br>bit 2: /TIMEPOLL<br>bit 3: /KEYPOLL<br>bit 4: inhibit timer<br>bit 5: enable status monitoring |
| 6Fh | BYTE | old status |
| 70h | WORD | signature DA34h |

**66-3 Bitfields for interrupts already grabbed by TAME:**

| | |
|---|---|
| bit 0 | INT 10h |
| bit 1 | INT 14h |
| bit 2 | INT 15h |
| bit 3 | INT 16h |
| bit 4 | INT 17h |
| bit 5 | INT 21h |
| bit 6 | INT 28h |

**66-4 Values for multitasker type:**

| | |
|---|---|
| 01h | DESQview |
| 02h | DoubleDOS |
| 03h | TopView |
| 04h | OmniView |
| 05h | VM/386 |

**66-5 Bitfields for type of task switching selected:**

| | |
|---|---|
| bit 0 | DESQview |
| bit 1 | DoubleDOS |
| bit 2 | TopView |
| bit 3 | OmniView |
| bit 4 | KeySwitch |
| bit 5 | HLT instruction |

## INT 21h Function 4Bh
## *INSTALLATION CHECK*

*ELRES v1.0 only*

Program: ELRES is an MS-DOS return code (errorlevel) recorder by David H. Bennett.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 4Bh | ES:BX -> ELRES history structure (see Function 2Bh Subfunction 454Ch) |
| DS:DX = 0000h:0000h | DX = DABEh (signature, DAve BEnnett) |

**Conflicts:** MS-DOS Function 4Bh (Chapter 24)
**See also:** Function 2Bh Subfunction 454Ch

---

### INT 21h Function 4Eh
### *INSTALLATION CHECK*

*WILDUNIX.COM internal*

**Program:** WILDUNIX.COM is a resident Unix-style wildcard expander by Steve Hosgood and Terry Barnaby.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 4Eh | AH = 99h if installed |
| DS:DX = 0000h:0000h | |

**Conflicts:** MS-DOS Function 4Eh (Chapter 24)

---

### INT 21h Function 5757h
### *Resident Manager - INSTALLATION CHECK*

*IBM Genie*
*(undocumented)*

**Program:** IBM Genie is a set of utility TSRs by Helix Software.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 5757h | AX = 0000h if installed |
| BX = 5757h | BX = *unknown* |
| | DX = *unknown* |
| | DS:SI -> list of 27 DWORD entry point addresses |

**Details:** Other functions are possible if BX <> 5757h, but details were not available at the time of writing.

---

### INT 21h Function 7734h
### *INSTALLATION CHECK*

*SCROLLit v1.7*
*(undocumented)*

**Program:** ScrollIt is a shareware backscroll utility by Bromfield Software Products.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 7734h | DX = 3477h if installed |
| | AX = segment of resident code |

---

### INT 21h Function 7761h
### *INSTALLATION CHECK*

*WATCH.COM v3.2+*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 7761h ('wa') | AX = 6177h |

**Details:** WATCH.COM is part of the "TSR" package by TurboPower Software.
**See also:** INT 16h Function 7761h

---

### INT 2Fh Function A4E0h
### *INSTALLATION CHECK*

*Futurus Team*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = A4E0h | AL = 52h ("R") if installed |
| | AH = major version plus 30h ("0") |
| | ES:BX -> *unknown* (*INT A4h handler*) |

**Details:** Older versions of Right Hand Man (from which Team evolved) store the signature "RH" at offset 103h in the INT 2Fh handler's segment.
**Conflicts:** none, but see Table 2-2
**See also:** Right Hand Man INT A4h (below)

---

### INT 2Fh Function AA00h
### *INSTALLATION CHECK*

*VIDCLOCK.COM*

**Program:** VIDCLOCK.COM is a memory-resident clock by Thomas G. Hanlin III.

**Registers at call:**
AX = AA00h

**Return Registers:**
AL = 00h not installed
FFh installed

**Conflicts:** none, but see Table 2-2

---

### INT 2Fh Function BE00h
### *INSTALLATION CHECK*
*REDVIEW*

**Program:** REDVIEW is a public-domain TSR by Alexandr Novy and Petr Horak which copies data sent to standard output to standard error when the former has been redirected to a file, thus allowing the data to be seen on the screen at the same time it is captured in a file.

**Registers at call:**
AX = BE00h

**Return Registers:**
AL = FFh if installed

**Conflicts:** none, but see Table 2-2

---

### INT 2Fh Function C000h
### *INSTALLATION CHECK*
*FN32 32 character filename utilities*

**Program:** FN32 is a TSR which supports 32 character filenames under PC/MS-DOS.

**Registers at call:**
AX = C000h

**Return Registers:**
AL = FFh if installed ES:DI -> signature string "FN32 32CHAR TSR"

**Details:** The TSR intercepts INT 21h calls and performs filename substitution by managing dictionary files in each directory which contains long filenames.

**Conflicts:** see Table 2-2

---

### INT 2Fh Function CAFEh
### *INSTALLATION CHECK*
*THELP v3.0*
*(undocumented)*

**Registers at call:**
AX = CAFEh
BX = 0000h

**Return Registers:**
BX = segment of resident code if installed, 0000h if not installed

**Conflicts:** TBSCANX (Chapter 59), FAXTSR (Chapter 64); see also Table 2-2

**See also:** TesSeRact Function 5453h (Chapter 51), AMIS INT 2Dh (Chapter 50)

---

### INT 2Fh Function EE00h
### *INSTALLATION CHECK*
*GRIDLOC.EXE*

**Program:** GRIDLOC is a PC security program by Intelligent Security Systems, Inc.

**Registers at call:**
AX = EE00h

**Return Registers:**
AL = FFh if installed

**Conflicts:** XVIEW (below), WEB (NI); see also Table 2-2

**See also:** Trusted Access INT 21h Function 40h (below)

---

### INT 2Fh Function F1h
### *INSTALLATION CHECK*
*MIN-MEM v2.11*
*(undocumented)*

**Program:** MIN-MEM is a shareware TSR manager by Biologic which permits up to 24 popup TSRs to be loaded but swapped out to disk, EMS, or XMS. One TSR at a time is brought back into memory at the user's request.

**Registers at call:**
AH = F1h
AL <> F1h

**Return Registers:**
AL = F1h if installed

**Conflicts:** DOS Extenders (Chapter 37); see also Table 2-2

---

### INT 60h
### *(NOT A VECTOR!) - INSTALLATION CHECK*
*Nabbit v2.0*

**Program:** Nabbit is a shareware resident screen data grabber by RSE Inc.

**Details:** Nabbit searches INT 60h through INT 66h for a vector which points at its ASCIZ signature string "iG" (69h 47h 00h), and uses the first free vector in that range if is it not already installed.

**Conflicts:** see Table 2-3

### INT 60h
### (NOT A VECTOR!) - INSTALLATION CHECK

*Buffit v3.0*
*(undocumented)*

**Program:** Buffit is a shareware scrollback utility by D.T. Hamilton.
**Details:** Buffit searches INT 60h through INT 6Fh for a vector which points at the ASCII signature, and uses the first free vector in that range if it is not already installed.

There is a private entry point immediately following the signature string, i.e. eight bytes beyond the address pointed at by the interrupt.
**Conflicts:** see Table 2-3

**66-6 Buffit private entry point calling convention:**

| Registers at call: | Return Registers: |
|---|---|
| AH = function | AH = new hotkey state (00h enabled, 01h disabled) |
| 00h get information and hotkey state | AL = hotkey scan code |
| 01h get information and toggle hotkey state | BH = hotkey shift states |
| | BL = *unknown* (01h) |
| | CX = segment of resident code |
| | DH = niterrupt number used for signature pointer |
| | DL = *unknown* (00h) |
| | SI = INT 09h handler offset |
| | DI = INT 21h handler offset |

### INT 67h
### INSTALLATION CHECK

*CUCKOO.COM*

**Program:** CUCKOO is a resident on-screen clock with optional hourly chime or cuckoo by an unknown author with revisions by Thomas A. Lundin.
**Details:** This is not a vector; when loaded for the first time, CUCKOO.COM uses the last unused (0000h:0000h) vector in the range 60h-67h to store the signature value 434Ch:4F4Bh ('CLOK').
**Conflicts:** EMS (Chapter 38); see also Table 2-3

### INT D0h
### NJFRERAM SIGNATURE VECTOR

*[not a vector!]*

**Program:** NJFRERAM is a resident free-memory display utility by Mike "Nifty James" Blaszczak.
**Details:** If NJFRERAM is installed, this vector points at the signature "NJ"
**Conflicts:** ROM BASIC (Chapter 54), APL*PLUS/PC (Chapter 53)

## APCAL

APCAL is an optionally-resident shareware appointment calendar by Gamma Software.

### INT 16h Function D724h, Subfunctions 00CBh and 00CCh
### GET UNKNOWN VALUES

*APCAL v3.20*
*(undocumented)*

| Registers at call: | Return Registers: |
|---|---|
| AX = D724h | AX = 0000h |
| CX = subfunction | BX = 0000h |
| 00CBh | DX:CX -> *unknown* or 0000h:0000h |
| 00CCh | DX:CX -> *unknown* (apparently an internal data area) |

**See also:** Function D724h Subfunction 00CDh

### INT 16h Function D724h, Subfunction 00CDh

*APCAL v3.20*
*(undocumented)*

***GET UNKNOWN VALUE***

**Registers at call:**
AX = D724h
CX = 00CDh

**Return Registers:**
AX = *unknown* (5345h seen)

**See also:** Function D724h Subfunctions 00CBh and 00CCh

## ASCII

ASCII is a popup ASCII table by Nick Zaikin, Jr.

### INT 2Fh Function 8000h

*ASCII v2.00*

***INSTALLATION CHECK***

**Registers at call:**
AX = 8000h
DS:SI -> 4-byte signature
  (11h 43h 53h 10h)

**Return Registers:**
AL = status
    01h installed, but signature did not match
    FFh installed, signature matches
        ES:DI -> resident copy of signature (may be followed by data)

**Details:** The interface used by this program appears to be a general-purpose interface which may be shared by multiple programs, so other programs may also support this call with differing signatures.

**Conflicts:** see Table 2-2

**See also:** Functions 8001h and 8002h

### INT 2Fh Function 8001h

*ASCII v2.00*

***INSTALLATION VERIFICATION***

**Registers at call:**
AX = 8001h
DS:SI -> ASCIZ signature
  string “ASCII 2.00 by
  Nick Zaikin Jr.”

**Return Registers:**
BX = result
    0000h if signature did not match
    *instance number* if signature matched
        ES:DI -> *unknown* (same as returned by Function 8000h)

**Details:** The interface used by this program appears to be a general-purpose interface which may be shared by multiple programs, so other programs may also support this call with differing signatures.

**Conflicts:** see Table 2-2

**See also:** Functions 8000h and 8002h

### INT 2Fh Function 8002h

*ASCII v2.00*

***UNDETERMINED FUNCTION***

**Registers at call:**
AX = 8002h
BX = *instance number* (value returned
  by Function 8001h)

**Return Registers:**
ES:DI -> *unknown* (same as returned by Function
  8000h) if instance number matches
call chained if instance number did not match

**Conflicts:** see Table 2-2

**See also:** Functions 8000h and 8001h

## AutoCAD

Autodesk’s AutoCAD (which retails for $3,500) is possibly the single most widely used full-fledged CAD program.

### INT 78h

*AutoCAD 10 ADI for Plotters*

***API***

**Registers at call:**
AX = function
    0001h initialize comm port and reset plotter
        BX = *unknown* (comm port?)

0002h close plotter
0003h move (pen up)
    BX = new X position
    CX = new Y position
0004h move (pen down)
    BX = new X position
    CX = new Y position
0005h select pen
    BX = pen number
0006h select drawing speed
    BX = drawing speed N
0007h select line style
    BX = line style N
0008h *return pen to pen carrier* (called before change-pen cmd)

**Return Registers:**
*unknown*

**Conflicts:** ADP-60 (Chapter 10), DBOS (Chapter 37), GO32 (Chapter 37)
**See also:** INT 7Ah, INT 7Bh

---

### INT 79h Function 0001h
### *DIGITIZER - INITIALIZE*

*AutoCAD Device Interface (undocumented)*

**Registers at call:**
AX = 0001h

**Return Registers:**
AX = 0001h *unknown self-test result*
BX = 0001h

**Conflicts:** ADP-60 (Chapter 10), AVATAR.SYS (Chapter 13), DBOS (Chapter 37), GO32 (Chapter 37)
**See also:** Functions 0002h and 0003h, INT 7Ah

---

### INT 79h Function 0002h
### *DIGITIZER - CLOSE*

*AutoCAD Device Interface (undocumented)*

**Registers at call:**
AX = 0002h

**Return Registers:**
none

**Conflicts:** ADP-60 (Chapter 10), AVATAR.SYS (Chapter 13), DBOS (Chapter 37), GO32 (Chapter 37)
**See also:** Functions 0001h and 0003h

---

### INT 79h Function 0003h
### *DIGITIZER - GET DIGITIZER STATUS*

*AutoCAD Device Interface (undocumented)*

**Registers at call:**
AX = 0003h

**Return Registers:**
AX = button state
    0002h no button pressed
        BX = X coordinate
        CX = Y coordinate
    0003h Pick button (0,A,B,C,D) pressed
        BX = X coordinate
        CX = Y coordinate
    0005h other button (1-9) pressed
        BX = number of button - 1
        CX = X coordinate
        DX = Y coordinate

**Details:** These return values are valid for the Houston Instruments HiPad Plus 9236 digitizer with a 16-button cursor.
**Conflicts:** ADP-60 (Chapter 10), AVATAR.SYS (Chapter 13), DBOS (Chapter 37), GO32 (Chapter 37)
**See also:** Functions 0001h and 0002h

---

**INT 7Ah** *AutoCAD Device Interface*
*API*

**Registers at call:**
*unknown*

**Return Registers:**
*unknown*

**Conflicts:** PRINDIR (Chapter 15), GO32 (Chapter 37), Novell NetWare (NI), Topware NOS (NI), X.PC Packet software (NI), IBM 3270 Workstation Program (NI)
**See also:** Function 0001h, INT 61h Function 0007h”OPTIMA”,AutoCAD INT 78h, INT 79h Function 0001h
**See also:** INT 7Bh

---

**INT 7Ah Function 0001h** *AutoCAD Device Interface*
***VIDEO DRIVER - GET ENTRY POINT***

**Registers at call:**
AX = 0001h

**Return Registers:**
AX = *unknown* (0003h for Compaq VGADI41.EXE)
BX = 0000h
DX:CX -> ADI entry point (see Table 66-7)

**Conflicts:** PRINDIR (Chapter 15), GO32 (Chapter 37), Novell NetWare (NI), Topware NOS (NI), X.PC Packet software (NI), IBM 3270 Workstation Program (NI)
**See also:** Function 07D0h

**66-7 ADI entry point calling convention:**

**Registers at call:**
STACK: DWORD -> request record (see Table 66-8)

**Return Registers:**
none

**66-8 Format of request record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | function number (00h-45h for Compaq VGADI41.EXE) |

---

**INT 7Ah Function 07D0h** *AutoCAD Device Interface*
***Compaq VGADI41.EXE - GET ENTRY POINT***

**Registers at call:**
AX = 07D0h

**Return Registers:**
AX = *unknown* (0003h)
BX = 0000h
DX:CX -> ADI entry point (see Table 66-7)

**Conflicts:** PRINDIR (Chapter 15), GO32 (Chapter 37), Novell NetWare (NI), Topware NOS (NI), X.PC Packet software (NI), IBM 3270 Workstation Program (NI)
**See also:** Function 0001h

---

**INT 7Bh** *AutoCAD ADI Callout (undocumented)*
***INTERFACE***

**Registers at call:**
AX = function
    0000h output blank line
    other *unknown*

**Return Registers:**
*unknown*

**Details:** This function is called by AutoCAD to perform its output.
**Conflicts:** GO32 (Chapter 37), Btrieve (Chapter 54), Eicon Access (NI)
**See also:** INT 78h, INT 7Ah

## CU Writer

CU Writer is a public-domain Thai-English word processor from Chulalongkorn University, Bangkok. The documentation provides only very sketchy information on the INT 10h API, as shown in Table 66-9.

**66-9 CU Writer functions:**

| *AH* | *Description* |
|---|---|
| 80h | Get option specified by AL |
| 81h | Set option |
| A0h | Set pixel write mode |
| B0h | Put pixel |
| B1h | Move to location |
| B2h | Draw line to location |
| B3h | Flood fill |
| B4h | Draw rectangle |
| B5h | Draw filled rectangle |
| C0h | Go to text row and column |
| C1h | Output text |
| C2h | Reverse text |
| C3h | Text bar |
| C4h | Text menu |
| D0h | Load picture |

**Conflicts:** DESQview (Chapter 44)

## FSBBS

The Fubar Systems BBS includes native support for Internet/UUCP mail. The prototype system is operated by sysop Christopher J. Ambler from San Luis Obispo, CA.

---

### INT F7h Function 00h

*FSBBS 2.0*

### *CONFIGURATION RECORD*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 00h | |
| AL = function | |
| 00h get configuration record | DS:DX -> configuration record |
| 01h set configuration record | |
| 02h get path for option<br>DS:DX -> option name | DS:DX -> path |
| 03h determine whether configuration record set | AX = status (0000h set, 0001h not yet set) |
| 04h get link state | AX = state (0000h unlinked, 0001h linked) |

**Details:** This information is preliminary and still subject to change. All of the INT F7h calls for FSBBS are used for interprogram communication between the BBS kernel and the programs it spawns.

**Conflicts:** DoubleDOS (Chapter 49)

**See also:** Function 01h

---

### INT F7h Function 01h

*FSBBS 2.0*

### *USER RECORD*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 01h | |
| AL = function | |
| 00h get user record for user currently online | DS:DX -> user record |
| 01h set user record<br>DS:DX -> user record | |

**Conflicts:** DoubleDOS (Chapter 49)

**See also:** Functions 00h and 02h

---

### INT F7h Function 02h

*FSBBS 2.0*

### *GET ACCOUNT NAME*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 02h | DS:DX -> 8-character blank-padded account name |

**Conflicts:** DoubleDOS (Chapter 49)

**See also:** Function 01h

### INT F7h Function 03h

***TERMINAL NUMBER***

*FSBBS 2.0*

| Registers at call: | Return Registers: |
|---|---|
| AH = 03h | |
| AL = function | |
| 00h get terminal index number | DX = index number |
| 01h set terminal index number | |
| DX = terminal index | |

**Conflicts:** DoubleDOS (Chapter 49)

### INT F7h Function 04h

***PASSDATA BUFFER***

*FSBBS 2.0*

| Registers at call: | Return Registers: |
|---|---|
| AH = 04h | none |
| AL = function | |
| 00h get PassData buffer contents | |
| DS:DX -> buffer for PassData contents | |
| 01h set PassData contents | |
| DS:DX -> buffer containing new PassData | |
| CH = length of data in buffer | |

**Conflicts:** DoubleDOS (Chapter 49)

### INT F7h Function 05h

***TIMER FUNCTIONS***

*FSBBS 2.0*

| Registers at call: | Return Registers: |
|---|---|
| AH = 05h | |
| AL = function | |
| 00h get time remaining | DX = number of minutes remaining |
| 01h get current time | DS:DX -> 8-character time string |
| 02h increment time | |
| DX = number of additional minutes | |
| 03h decrement time | |
| DX = number of minutes | |

**Conflicts:** DoubleDOS (Chapter 49)

**See also:** Function 06h

### INT F7h Function 06h

***FUNCTION AVAILABILITY***

*FSBBS 2.0*

| Registers at call: | Return Registers: |
|---|---|
| AH = 06h | *unknown* |
| AL = function | |
| 00h determine whether function is available | |
| DX = index of function | |
| 01h set function availability | |
| DX = index of function | |
| *unknown* | |

**Conflicts:** DoubleDOS (Chapter 49)

**See also:** Functions 05h and 07h

---

### INT F7h Function 07h
*FSBBS 2.0*

***DUMP FUNCTIONS***

**Registers at call:**
AH = 07h
AL = function
    00h get current dump mode
    01h set dump mode
        DL = mode

**Return Registers:**

DL = mode (for 00h)

**Conflicts:** DoubleDOS (Chapter 49)
**See also:** Function 06h

## GOLD

GOLD is a TSR by Bob Eager which makes the NumLock key return the code for F1; the purpose is to improve Kermit's VTxxx emulation. As with all programs using multiplex numbers in the range C0h to FFh, there are numerous potential conflicts which are listed in Table 2-2.

---

### INT 2Fh Function DC00h
*GOLD.COM*

***INSTALLATION CHECK***

**Registers at call:**
AX = DC00h

**Return Registers:**
AL = status (00h not installed, FFh installed)

---

### INT 2Fh Function DC01h
*GOLD.COM*

***GET STATE***

**Registers at call:**
AX = DC01h

**Return Registers:**
AL = status (00h off, 01h on)

**See also:** Functions DC00h and DC02h

---

### INT 2Fh Function DC02h
*GOLD.COM*

***SET STATE***

**Registers at call:**
AX = DC02h
DL = new state (00h off, 01h on)

**Return Registers:**
AL = 00h (OK)

**See also:** Function DC01h

## Jot-It!

As with all programs using multiplex numbers in the range C0h to FFh, there are numerous potential conflicts which are listed in Table 2-2.

---

### INT 2Fh Function FB00h
*Jot-It! v1.50*
*(undocumented)*

***INSTALLATION CHECK***

**Registers at call:**
AX = FB00h

**Return Registers:**
AX = FFFFh if installed
    BX = version (BCD, BH=major, BL=minor)

**See also:** Functions FB01h and FB03h

---

### INT 2Fh Function FB01h
*Jot-It! v1.50*
*(undocumented)*

***GET USER NAME***

**Registers at call:**
AX = FB01h

**Return Registers:**
DX:BX -> ASCIZ user name

**See also:** Function FB02h

### INT 2Fh Function FB02h
*Jot-It! v1.50*
*(undocumented)*

***GET MESSAGE DIRECTORY***

**Registers at call:**
AX = FB02h

**Return Registers:**
DX:BX -> ASCIZ name of directory in which messages are stored

**See also:** Function FB01h

### INT 2Fh Function FB03h
*Jot-It! v1.50*
*(undocumented)*

***UNINSTALL***

**Registers at call:**
AX = FB03h

**Return Registers:**
resident code removed from memory

**Details:** CAUTION: No checks are performed to ensure that the interrupt vectors being unhooked (08h,09h,28h,2Fh) actually point at the Jot-It! code.

**See also:** Function FB00h

## Kingswood TSR Windows

### INT 2Fh Function 3901h, Subfunction 02h
*Kingswood TSR Windows*

***OPEN WINDOW***

**Registers at call:**
AX = 3901h
BL = 02h

**Return Registers:**
AX = error code (0000h if successful)

**Details:** This function opens the next TSR window on top of any others. Only three TSR windows can be opened at any one time. The three windows are all 40x11 characters, partly overlapping.

**See also:** Function 3901h Subfunctions 03h, 05h, and 06h; Function 39h Subfunction 00h (Chapter 51)

### INT 2Fh Function 3901h, Subfunction 03h
*Kingswood TSR Windows*

***HIDE WINDOWS***

**Registers at call:**
AX = 3901h
BL = 03h

**Return Registers:**
AX = error code (0000h if successful)

**Details:** This function hides any visible TSR windows from view.

**See also:** Function 3901h Subfunctions 02h and 05h; Function 39h Subfunction 00h (Chapter 51)

### INT 2Fh Function 3901h, Subfunction 04h
*Kingswood TSR Windows*

***SHOW WINDOWS***

**Registers at call:**
AX = 3901h
BL = 04h

**Return Registers:**
AX = error code (0000h if successful)

**Details:** This function re-displays all TSR windows after a Hide Windows call.

**See also:** Function 3901h Subfunctions 02h and 03h, Function 39h Subfunction 00h (Chapter 51)

### INT 2Fh Function 3901h, Subfunction 05h
*Kingswood TSR Windows*

***CLOSE WINDOW***

**Registers at call:**
AX = 3901h
BL = 05h

**Return Registers:**
AX = error code (0000h if successful)

**Details:** This function closes the last-opened TSR window.

**See also:** Function 3901h Subfunction 02h, Function 39h Subfunction 00h (Chapter 51)

### INT 2Fh Function 3901h, Subfunction 06h
*Kingswood TSR Windows*

***SET WINDOW TITLE***

**Registers at call:**
AX = 3901h

BL = 06h
DS:SI -> title string

**Return Registers:**
AX = error code (0000h if successful)

**See also:** Function 3901h Subfunction 02h, Function 39h Subfunction 00h (Chapter 51)

---

### INT 2Fh Function 3901h, Subfunction 07h
*Kingswood TSR Windows*

### *POSITION CURSOR*

**Registers at call:**
AX = 3901h
BL = 07h
CH = Y coordinate (0-10)
CL = X coordinate (0-39)

**Return Registers:**
AX = error code (0000h if successful)

**Details:** The hardware cursor is always disabled when a TSR window is opened; this call only sets a text position.

**See also:** Function 3901h Subfunctions 08h and 09h, Function 39h Subfunction 00h (Chapter 51)

---

### INT 2Fh Function 3901h, Subfunction 08h
*Kingswood TSR Windows*

### *DISPLAY STRING*

**Registers at call:**
AX = 3901h
BL = 08h
DS:SI -> string

**Return Registers:**
AX = error code (0000h if successful)

**Details:** The text is not clipped. This routine understands Tab, NewLine and Carriage Return.

**See also:** Function 3901h Subfunction 07h, Function 39h Subfunction 00h (Chapter 51)

---

### INT 2Fh Function 3901h, Subfunction 09h
*Kingswood TSR Windows*

### *SCROLL WINDOW*

**Registers at call:**
AX = 3901h
BL = 09h
CL = direction: 00h up, FFh down, 00h clear window

**Return Registers:**
AX = error code (0000h if successful)

**See also:** Function 3901h Subfunction 07h, Function 39h Subfunction 00h (Chapter 51)

---

### INT 2Fh Function 3901h, Subfunction 0Ah
*Kingswood TSR Windows*

### *SOUND BEEPER*

**Registers at call:**
AX = 3901h
BL = 0Ah
DX = sound divisor, or 0 for silence
(divide 1843200 by desired frequency to get value)
CL = sound length in 18.2 Hz clock ticks

**Return Registers:**
AX = error code (0000h if successful)

**See also:** Function 39h Subfunction 00h (Chapter 51)

---

### INT 2Fh Function 3901h, Subfunction 0Bh
*Kingswood TSR Windows*

### *ADD OR REMOVE USER*

**Registers at call:**
AX = 3901h
BL = 0Bh
CL = increment: +1 if adding a new user, -1 if removing a user

**Return Registers:**
AX = error code (0000h if successful)

**Details:** The TSR windows resident module may only be removed when the internal user count is zero.

**See also:** Function 3901h Subfunction 02h, Function 39h Subfunction 00h (Chapter 51)

## METZTSR

METZTSR.COM prevents METZ applications (such as the MAGIC screen saver) inactivity timeout while running a DOSapp under MS Windows.

The default multiplex number for METZTSR is A9h, but may be set to any value from 80h to FFh with a commandline switch. Like all programs using multiplex numbers in this range, there are several potential conflicts which are listed in Table 2-2.

---

**INT 2Fh Function A900h** *METZTSR.COM*

***INSTALLATION CHECK***

**Registers at call:**
AX = A900h
CF set

**Return Registers:**
CF clear if resident
AX = 97FFh
CF set if not present

**See also:** Functions A901h and A902h

---

**INT 2Fh Function A901h** *METZTSR.COM*

***GET TIME OF LAST KEYBOARD ACTIVITY***

**Registers at call:**
AX = A901h
CF set

**Return Registers:**
CF clear if successful
AX:DX = BIOS time at which INT 09h was last invoked
CF set if not present

**See also:** IRQ1 INT 09h (Chapter 3), BIOS INT 1Ah Function 00h (Chapter 4)

---

**INT 2Fh Function A902h** *METZTSR.COM*

***SET METZ Ctrl-Alt-Del FLAG***

**Registers at call:**
AX = A902h
BL = new value
00h Ctrl-Alt-Del not allowed
else Ctrl-Alt-Del allowed (startup default is 01h)
CF set

**Return Registers:**
CF clear if successful
AX = 97FFh
CF set if not resident

**See also:** Function A903h

---

**INT 2Fh Function A903h** *METZTSR.COM*

***GET METZ Ctrl-Alt-Del FLAG***

**Registers at call:**
AX = A903h
CF set

**Return Registers:**
CF clear if successful
AX = 97FFh if Ctrl-Alt-Del allowed
AX = 0000h if Ctrl-Alt-Del not allowed
CF set if not resident

**See also:** Function A902h

## PCL-838

PCL-838 is a resident utility for controlling a stepmotor adapter board by Advantec.

---

**INT 2Fh Function D200h** *PCL-838.EXE*

***INSTALLATION CHECK***

**Registers at call:**
AX = D200h

**Return Registers:**
AL = FFh *if installed* (documented as AX = return value)

**Conflicts:** Quarterdeck RPCI (Chapter 63); see also Table 2-2

**See also:** Functions D201h and D202h

---

**INT 2Fh Function D201h** *PCL-838.EXE*

***UNINSTALL***

**Registers at call:**
AX = D201h

**Return Registers:**
AX = return value

**Conflicts:** Quarterdeck RPCI (Chapter 63); see also Table 2-2

**See also:** Functions D200h and D202h

**INT 2Fh Function D202h** *PCL-838.EXE*

***EXECUTE PCL838 COMMANDS***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = D202h | AX = return value |
| CX:BX -> parameter table | |

**Conflicts:** Quarterdeck RPCI (Chapter 63); see also Table 2-2

**See also:** Functions D200h and D201h

## PC Thuis Shell and Vanderaart Text Windows

The PC Thuis Organizer Shell was written by John Vanderaart and published in the June/July 1990 issue of PC Thuis Power magazine.

All of the calls in this section conflict with relocated IRQs for DESQview (Chapter 44), OS/2 (Chapter 26), and IBM 3270 emulation (NI); and TIL Expert AIM (NI). Further, the two programs conflict with each other on many calls.

**INT 50h Function 0000h** *Vanderaart TEXT WINDOWS, PC Thuis Shell*

***OPEN TEXT WINDOW***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 0000h | AX = window handle or |
| ES:BX -> name string or ES:0000h if none | 0000h if not installed |
| CH,CL = row,column of upper left corner | FFFFh on error |
| DH,DL = row,column of lower right corner | |

**See also:** Functions 0001h and 0002h

**INT 50h Function 0001h** *Vanderaart TEXT WINDOWS, PC Thuis Shell*

***CLOSE TEXT WINDOW***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 0001h | none |
| DI = window handle | |

**See also:** Function 0000h

**INT 50h Function 0002h** *Vanderaart TEXT WINDOWS*

***PUT CHARACTER IN WINDOW***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 0002h | AX = status |
| BL = character | 0000h if successful |
| BH = attribute | FFFFh if outside window |
| DH,DL = row,column | |
| DI = window handle | |

**See also:** Function 0000h

**INT 50h Function 0002h** *PC Thuis Organizer Shell*

***PLOT TEXT***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 0002h | AX = status |
| ES:BX -> text string | 0000h successful (text fits in window) |
| DH,DL = row,column of upper left corner | FFFFh error |
| DI = window handle | |

**INT 50h Function 0003h** *Vanderaart TEXT WINDOWS*

***OUTPUT LINE TO WINDOW***

**Registers at call:**

AX = 0003h

ES:BX -> text string
CX = string length (0000h if ASCIZ string)
DL = position (FFh centered, else flush left)
DH = starting row
DI = window handle

**Return Registers:**
AX = status
0000h successful
FFFFh did not fit in window

---

### INT 50h Function 0003h
*PC Thuis Organizer Shell*

***WRITE FILE***

**Registers at call:**
AX = 0003h
ES:BX -> data to be written
CX = number of bytes to write
DS:SI -> filename

**Return Registers:**
AX = status
0000h successful
FFFFh error

**See also:** PC Thuis Shell Function 0004h

---

### INT 50h Function 0004h
*Vanderaart TEXT WINDOWS*

***GET KEY***

**Registers at call:**
AX = 0004h
CH = type
  00h any key
  01h 'J' or 'N' (Dutch for yes/no)

**Return Registers:**
AX = key

**See also:** BIOS INT 16h Function 00h (Chapter 4)

---

### INT 50h Function 0004h
*PC Thuis Organizer Shell*

***READ FILE***

**Registers at call:**
AX = 0004h
ES:BX -> buffer for data
CX = number of bytes to read or 0000h for entire file
DL = file type (01h setting shell, 02h setting sterm, 03h INT21h file)
DS:SI -> filename

**Return Registers:**
AX = status
0000h successful
FFFFh error

**Details:** File type numbers are maintained by John Vanderaart; if a new file type is needed, a type number should be requested from him through the magazine: PC Thuis BV, Spaarne 55, 2011 CE HAARLEM, The Netherlands.

**See also:** Function 0003h

---

### INT 50h Function 0005h
*Vanderaart TEXT WINDOWS*

***CHANGE ATTRIBUTE***

**Registers at call:**
AX = 0005h
BL = new attribute
CH,CL = row,column of upper left corner
DH,DL = row,column of lower right corner
DI = window handle

**Return Registers:**
none

---

### INT 50h Function 0005h
*PC Thuis Organizer Shell*

***PROMPT YES/NO***

**Registers at call:**
AX = 0005h
ES:BX -> prompt string (ES:0000h if no prompt)

**Return Registers:**
AX = key pressed
0000h "J" (Dutch "Ja" = "Yes")
FFFFh "N" (Dutch "Nee" = "No")

**See also:** PC Thuis Shell Function 0008h

---

## INT 50h Function 0006h

*Vanderaart TEXT WINDOWS*

### *EDIT LINE IN WINDOW*

**Registers at call:**
AX = 0006h
ES:BX -> text string
CH = type of input (see below)
DH,DL = row,column of upper left corner
DI = window handle

**Return Registers:**
AX = key which terminated entry
0000h Enter
0001h Esc
0002h Down arrow
0003h Up arrow
0004h F10

**66-10 Values for type input:**

| | |
|---|---|
| 00h | everything |
| 01h | uppercase only |
| 02h | positive numbers |
| 03h | Dutch postal code (“9999 AA”) |
| 04h | ’J’ or ’N’ (Dutch yes/no) |
| 05h | telephone or FAX number |
| 06h | positive or negative number |
| 07h | date (dd/mm/yy) |
| 08h | money |
| 09h | ’1’ through ’8’ |
| 0Ah | ’1’ through ’4’ |
| 0Bh | uppercase filenames |

---

## INT 50h Function 0006h

*PC Thuis Organizer Shell*

### *ALERT USER*

**Registers at call:**
AX = 0006h
ES:BX -> string

**Return Registers:**
none

---

## INT 50h Function 0007h

*PC Thuis Organizer Shell*

### *DO LINE*

**Registers at call:**
AX = 0007h
ES:BX -> text string
CX = string length in bytes (0000h if NUL-terminated)
DL = FFh to center string, else flush left
DH = upper left row
DI = window handle

**Return Registers:**
AX = status
0000h successful
FFFFh error

**See also:** Function 0008h

---

## INT 50h Function 0008h

*PC Thuis Organizer Shell*

### *DO MENU*

**Registers at call:**
AX = 0008h
ES:BX -> menu structure

**Return Registers:**
AL = index 1 or FFh if not selected
AH = index 2 or FFh if not selected
BL = index 3 or FFh if not selected
BH = index 4 or FFh if not selected

**See also:** Functions 0005h, 0007h, and 000Ch

---

## INT 50h Function 0009h

*PC Thuis Organizer Shell*

### *MESSAGE ON*

**Registers at call:**
AX = 0009h
ES:BX -> message string

**Return Registers:**
none

**See also:** Function 000Ah

---

## INT 50h Function 000Ah

*PC Thuis Organizer Shell*

### *MESSAGE OFF*

**Registers at call:**
AX = 000Ah

**Return Registers:**
none

**See also:** Function 0009h

---

## INT 50h Function 000Bh

*PC Thuis Organizer Shell*

### *CHANGE ATTRIBUTE*

**Registers at call:**
AX = 000Bh
BL = new attribute
CH,CL = row,column of upper left corner
DH,DL = row,column of lower right corner
DI = window handle

**Return Registers:**
none

---

## INT 50h Function 000Ch

*PC Thuis Organizer Shell*

### *DO REQUEST*

**Registers at call:**
AX = 000Ch
ES:BX -> request structure

**Return Registers:**
AX = status (0000h confirmed, FFFFh denied)

**See also:** Function 0008h

---

## INT 50h Function 000Dh

*PC Thuis Organizer Shell*

### *EDIT LINE*

**Registers at call:**
AX = 000Dh
ES:BX -> text string
CL = length
CH = type
    bit 0: force uppercase
    bit 1: integer
    bit 2: no spaces allowed
    bit 3: no cursor keys
DH,DL = row,column of upper left corner
DI = window handle

**Return Registers:**
AX = result code

---

## INT 50h Function 000Eh

*PC Thuis Organizer Shell*

### *PLOT CHARACTER*

**Registers at call:**
AX = 000Eh
BL = character
BH = attribute
DH,DL = row,column at which to plot
DI = window handle

**Return Registers:**
AX = status
    0000h successful
    FFFFh errror

## INT 50h Function 000Fh

*PC Thuis Organizer Shell*

### *EMPTY WINDOW*

**Registers at call:**
AX = 000Fh
BL = character
BH = attribute
DI = window handle

**Return Registers:**
none

---

## INT 50h Function 0010h

*PC Thuis Organizer Shell*

### *TRACE MENU*

**Registers at call:**
AX = 0010h
ES:BX -> first menu structure
CL = hotkey to look up

**Return Registers:**
AL = index 1 or FFh if not selected
AH = index 2 or FFh if not selected
BL = index 3 or FFh if not selected
BH = index 4 or FFh if not selected

---

## INT 50h Function 0011h

*PC Thuis Organizer Shell*

### *MOVE MEMORY*

**Registers at call:**
AX = 0011h
DS:SI -> source
ES:DI -> destination
CX = number of bytes to move (0000h = *until NUL string terminator*)

**Return Registers:**
none

**See also:** Function 0012h

---

## INT 50h Function 0012h

*PC Thuis Organizer Shell*

### *COMPARE MEMORY*

**Registers at call:**
AX = 0012h
DS:SI -> source
ES:DI -> destination
CX = number of bytes to compare (0000h = *until NUL string terminator*)

**Return Registers:**
AX = status
    0000h same
    FFFFh different

**See also:** Function 0011h

---

## INT 50h Function 0013h

*PC Thuis Organizer Shell*

### *GET KEY*

**Registers at call:**
AX = 0013h
CH = type flags
    bit 0: force uppercase
    bit 1: integer
    bit 2: no spaces

**Return Registers:**
AX = keystroke

---

## INT 50h Function 0014h

*PC Thuis Organizer Shell*

### *SCROLL WINDOW*

**Registers at call:**
AX = 0014h
BL = direction (06h up, 07h down)
BH = attribute
DI = window handle

**Return Registers:**
none

**See also:** BIOS INT 10h Functions 06h and 07h (Chapter 8)

*PC Thuis Organizer Shell*

### INT 50h Function 0015h
### *GET MEMORY HANDLE*

**Registers at call:**
AX = 0015h
BL = handle size
    00h 65536 bytes (64K)
    01h 65535 bytes (64K-1)
    02h 32768 bytes (32K)
    03h 32767 bytes (32K-1)

**Return Registers:**
AX = segment

**See also:** INT 21h Function 48h

## PCW Weather Card

*PCW Weather Card interface*

### INT 21h Function 7070h, Subfunction 6060h
### *GET DATA SEGMENT*

**Registers at call:**
AX = 7070h
BX = 6060h
CX = DX = SI = DI = 7070h

**Return Registers:**
AX = segment of data structure

**Details:** The data structure is at offset 516 from this segment.

The update byte is at offset 514 from this segment; updates are once per second while this byte is nonzero and it is decremented once per second. While this byte is 0 updates are once per minute.

**See also:** Function 7070h Subfunction 7070h

**66-11 Format of data structure:**

| *Offset* | *Type* | *Description* |
|---|---|---|
| 00h | WORD | hour |
| 02h | WORD | minute |
| 04h | WORD | second |
| 06h | WORD | day |
| 08h | WORD | month |
| 0Ah | WORD | year |
| 0Ch | WORD | *unknown* |
| 0Eh | WORD | relative barometric pressure (in 1/100 inches) |
| 10h | WORD | *unknown* |
| 12h | WORD | *unknown* |
| 14h | WORD | temperature 1 (in 1/10 degrees F) |
| 16h | WORD | temperature 1 lowest (in 1/10 degrees F) |
| 18h | WORD | temperature 1 highest (in 1/10 degrees F) |
| 1Ah | WORD | temperature 2 (in 1/10 degrees F) |
| 1Ch | WORD | temperature 2 lowest (in 1/10 degrees F) |
| 1Eh | WORD | temperature 2 highest (in 1/10 degrees F) |
| 20h | WORD | wind speed (in MPH) |
| 22h | WORD | average of 60 wind speed samples (in MPH) |
| 24h | WORD | highest wind speed (in MPH) |
| 26h | WORD | wind chill factor (in 1/10 degrees F) |
| 28h | WORD | lowest wind chill factor (in 1/10 degrees F) |
| 2Ah | WORD | *unknown* |
| 2Ch | WORD | wind direction (in degrees) |
| 2Eh | WORD | accumulated daily rainfall (in 1/10 inches) |
| 30h | WORD | accumulated annual rainfall (in 1/10 inches) |

---

### INT 21h Function 7070h, Subfunction 7070h
### *INSTALLATION CHECK*

*PCW Weather Card interface*

**Registers at call:**
AX = 7070h
BX = 7070h
CX = DX = SI = DI = 7070h

**Return Registers:**
AX = 0070h
BX = 0070h
CX = DX = SI = DI = 0070h

**See also:** Function 7070h Subfunction 6060h, Function 8080h

---

### INT 21h Function 8080h
### *UNINSTALL PCW.COM AND FREE MEMORY*

*PCW Weather Card interface*

**Registers at call:**
AX = 8080h

**Return Registers:**
*unknown*

**Conflicts:** European MS-DOS (Chapter 25)
**See also:** Function 7070h Subfunction 7070h

## RAID

RAID (Resident AID) is a TSR utility program by Ross Neilson Wentworth that resides mostly in EMS. Although there are no direct conflicts, Table 2-2 lists a number of potential conflicts with other programs using multiplex numbers in the range 80h to FFh.

---

### INT 2Fh Function 9000h
### *INSTALLATION CHECK*

*RAID*
*(undocumented)*

**Registers at call:**
AX = 9000h

**Return Registers:**
AL = FFh if installed

---

### INT 2Fh Function 9001h
### *GET UNKNOWN VALUE*

*RAID*
*(undocumented)*

**Registers at call:**
AX = 9001h

**Return Registers:**
DX:AX -> *unknown*

**See also:** Function 9000h

---

### INT 2Fh Function 9002h
### *GET RESIDENT SEGMENT*

*RAID*
*(undocumented)*

**Registers at call:**
AX = 9002h

**Return Registers:**
AX = segment of resident (conventional memory) portion

**See also:** Function 9000h

---

### INT 2Fh Function 9003h
### *UNINSTALL*

*RAID*
*(undocumented)*

**Registers at call:**
AX = 9003h

**Return Registers:**
*unknown*

**See also:** Function 9000h

---

### INT 2Fh Function 9004h
### *GET UNKNOWN VALUE*

*RAID*
*(undocumented)*

**Registers at call:**
AX = 9004h

**Return Registers:**
AX = *first available paragraph past end of resident portion*
CX destroyed

**See also:** Function 9000h

## RighTime

RighTime is a TSR by G.T. Becker which continuously adjusts the system time to correct for clock drift.

---

**INT 1Ah Function F7h** *RighTime v1.1*

***TEMPORARILY DISABLE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = F7h | none |

**Details:** Any AH value from F0h-F7h or F9h-FEh will perform this function in version 1.1, but F7h is the function called by the transient portion.

**See also:** Functions F8h and FFh

---

**INT 1Ah Function F8h** *RighTime v1.1*

***ENABLE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = F8h | none |

**Details:** RighTime is TeSseRact-compatible (see INT 2Fh Function 5453h in Chapter 51) and modifies its TeSseRact program identifier based on its current state: "RighTime" when enabled, "RighTim"F7h when disabled.

**See also:** Functions F7h and FFh

---

**INT 1Ah Function FFh** *RighTime v1.1*

***PERMANENTLY DISABLE***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = FFh | none |

**Details:** Upon being permanently disabled, RighTime closes the file handle referencing its executable (which is updated with time correction information every two minutes while RighTime is enabled).

**Conflicts:** AT&T 6300 (Chapter 7)

## Screen Thief

---

**INT 2Dh Function 10h** *Screen Thief v1.00 (undocumented)*

***FREE HIGH MEMORY BUFFERS***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AL = 10h | none |
| AH = AMIS multiplex number for Screen Thief | |

**Details:** This function releases any code and data stored in EMS, DOS UMBs, or XMS UMBs, but does not release the low-memory stub; this may be used to effect a partial uninstall if INT 2Dh Function 02h fails.

**See also:** INT D8h, AMIS INT 2Dh Function 02h (Chapter 50)

---

**INT D8h through INT DFh** *Screen Thief v1.00*

***RELOCATED IRQ0 THROUGH IRQ7***

**Purpose:** By reprogramming the interrupt controller to use these interrupts, Screen Thief is able to intercept interrupts before even programs that completely override the normal hardware interrupt handlers.

**Details:** Screen Thief relocates IRQs 0 through 7 to INT D8h to INT DFh by default, but may be directed via a commandline switch to use any range starting at a multiple of 8 between 78h and E0h.

**Conflicts:** Victor 9000 INT DFh (Chapter 7), APL*PLUS/PC (Chapter 53), ROM BASIC (Chapter 54), PC/370 INT DCh (below)

**See also:** Screen Thief INT 2Dh Subfunction 10h, Hardware INT 08h (Chapter 3), DESQview INT 50h (Chapter 44)

## SETDRVER

SETDRVER is a public domain TSR by Jacob Rieper which sets the apparent DOS version analogously to MS-DOS SETVER. The SETDRVER API conflicts with StuffIt (Chapter 13), and potentially conflicts with numerous other programs using multiplex numbers in the range C0h to FFh, which are listed in Table 2-2.

---

**INT 2Fh Function E000h** *SETDRVER.COM v2.10+*

***INSTALLATION CHECK***

| Registers at call: | Return Registers: |
|---|---|
| AX = E000h | AX = 4A52h ("JR") if present |

**Details:** This installation check differs from the usual one of returning AL=FFh.

**See also:** Function E001h, MS-DOS INT 21h Function 52h (Chapter 24)

---

**INT 2Fh Function E001h** *SETDRVER.COM v2.10+*

***GET SETDRVER VERSION***

| Registers at call: | Return Registers: |
|---|---|
| AX = E001h | AH = major version<br>AL = minor version |

**See also:** Function E000h

---

**INT 2Fh Function E002h** *SETDRVER.COM v2.10+*

***GET ORIGINAL DOS VERSION INFO***

| Registers at call: | Return Registers: |
|---|---|
| AX = E002h | AL = FFh if successful<br>BH = major DOS version<br>BL = minor DOS version<br>CH = DOS version flag<br>CL = OEM number<br>DH = major DR-DOS version number (FFh if unknown)<br>DL = minor DR-DOS version number (FFh if unknown) |

**See also:** Functions E003h and E007h, MS-DOS INT 21h Function 30h (Chapter 24)

---

**INT 2Fh Function E003h** *SETDRVER.COM v2.10+*

***RESET INTERNAL VARIABLES***

| Registers at call: | Return Registers: |
|---|---|
| AX = E003h<br>BH = new major DOS version<br>BL = new minor DOS version<br>CH = new DOS version flag<br>CL = new DOS revision number<br>DH = new OEM number | none |

**See also:** Function E002h

---

**INT 2Fh Functions E004h and E005h** *SETDRVER.COM v2.10+*

***ENABLE/DISABLE TSR***

| Registers at call: | Return Registers: |
|---|---|
| AX = E004h enable<br>AX = E005h disable | AL = FFh if successful |

**See also:** Functions E000h and E006h

---

### INT 2Fh Function E006h

*SETDRVER.COM v2.10+*

***GET TSR STATUS***

**Registers at call:**
AX = E006h

**Return Registers:**
AL = FFh if successful
BL = status
01h resident and active
02h resident and inactive

---

### INT 2Fh Function E007h

*SETDRVER.COM v2.10+*

***GET TaskMAX STATUS AT INSTALLATION***

**Registers at call:**
AX = E007h

**Return Registers:**
AL = FFh if successful
BL = status
00h if TaskMAX not loaded before SETDRVER
FFh if TaskMAX was loaded before SETDRVER

**See also:** Function E003h

---

### INT 2Fh Function E0h

*SETDRVER.COM*

***RESERVED FOR FUTURE USE***

**Registers at call:**
AH = E0h
AL = 08h-10h

**Return Registers:**
none

## SitBack

SitBack is a background file backup utility by SitBack Technologies, Inc. which initiates backups whenever the system is idle. The SitBack API conflicts with relocated IRQs for DESQview (Chapter 44) and DoubleDOS (Chapter 49), the Cluster Adapter (NI), NetBIOS (NI), the Microsoft Network Transport Layer (NI), Alloy NTNX (NI), and ISOLAN Multi-Protocol Software (NI).

---

### INT 5Bh Function 5254h, Subfunction 04h

*SitBack v3.02R (undocumented)*

***GET UNKNOWN VALUE***

**Registers at call:**
AX = 5254h
DL = 04h

**Return Registers:**
ES:BX -> *unknown* in resident portion

**See also:** Function 8485h Subfunctions 71h and 72h

---

### INT 5Bh Function 8485h, Subfunction 70h

*SitBack v3.02R (undocumented)*

***INSTALLATION CHECK***

**Registers at call:**
AX = 8485h
DL = 70h

**Return Registers:**
CX = 8485h if installed
DX:AX -> *unknown* (configuration data?)

**See also:** Function 5254h Subfunction 04h, Function 8485h Subfunction 78h

---

### INT 5Bh Function 8485h, Subfunction 71h

*SitBack v3.02R (undocumented)*

***SET UNKNOWN FLAG AND GET UNKNOWN ADDRESS***

**Registers at call:**
AX = 8485h
DL = 71h

**Return Registers:**
ES:BX -> FAR entry point to *unknown*

**Details:** The flag which is modified is located at the address returned by Function 5254h Subfunction 04h.

**See also:** Function 8485h Subfunction 72h

---

### INT 5Bh Function 8485h, Subfunction 72h
*SitBack v3.02R (undocumented)*

### *CLEAR UNKNOWN FLAG*

**Registers at call:**
AX = 8485h
DL = 72h

**Return Registers:**
none

**Details:** The flag which is modified is located at the address returned by Function 5254h Subfunction 04h.

**See also:** Function 8485h Subfunction 71h

---

### INT 5Bh Function 8485h, Subfunctions 73h to 76h
*SitBack v3.02R (undocumented)*

### *UNDETERMINED FUNCTIONS*

**Registers at call:**
AX = 8485h
DL = 73h to 76h
CX = *unknown* (Functions 75h,76h only)
*others, if any, unknown*

**Return Registers:**
*unknown*

**Details:** Subfunction 76h conditionally calls the code for Subfunction 75h.

---

### INT 5Bh Function 8485h, Subfunction 77h
*SitBack v3.02R (undocumented)*

### *SET UNKNOWN FLAG*

**Registers at call:**
AX = 8485h
DL = 77h

**Return Registers:**
none

---

### INT 5Bh Function 8485h, Subfunction 78h
*SitBack v3.02R (undocumented)*

### *GET RESIDENT DATA SEGMENT*

**Registers at call:**
AX = 8485h
DL = 78h

**Return Registers:**
CX = 5342h if supported
ES = AX = segment of TSR data

**See also:** Function 8485h Subfunctions 70h and 79h

---

### INT 5Bh Function 8485h, Subfunction 79h
*SitBack v3.02R (undocumented)*

### *GET DTA*

**Registers at call:**
AX = 8485h
DL = 79h

**Return Registers:**
CX = 5342h if supported
ES:BX -> DTA set by last INT 21h Function 1Ah

**Details:** This function is provided by SBOS.EXE rather than SB.EXE.

**See also:** INT 21h Function 1Ah

---

### INT 5Bh Function 8485h, Subfunction 7Ah
*SitBack v3.02R (undocumented)*

### *TOGGLE UNKNOWN FLAG*

**Registers at call:**
AX = 8485h
DL = 7Ah

**Return Registers:**
CX = 5342h if supported
AL = new value of *unknown* (00h or 01h)

## StackMan

StackMan is a freeware stack manager by Franz Veldman of ESaSS B.V. which functions as a replacement for the DOS STACKS= command as well as permitting multiple TSRs to share a pool of stack space.

The installation check consists of testing for the string "STACKXXX" at offset 0Ah from the INT B4h handler.

---

### INT 2Fh Function C9FFh

*StackMan Callout*

### *INSTALLATION BROADCAST*

**Registers at call:**
AX = C9FFh
BL = BCD version number
CX = number of stacks
DX = stack size in bytes

**Return Registers:**
none

**Details:** This function is called by StackMan when it goes resident to inform interested TSRs that its API is available.

**Conflicts:** see Table 2-2

**See also:** INT B4h, INT B5h

---

### INT B4h

*StackMan*

### *REQUEST NEW STACK*

**Registers at call:**
none

**Return Registers:**
SS:SP -> new stack

**Conflicts:** ROM BASIC (Chapter 54)

**See also:** INT 2Fh Function C9FFh, INT B5h

---

### INT B5h

*StackMan*

### *RESTORE ORIGINAL STACK*

**Registers at call:**
SS:SP -> stack returned by INT B4h

**Return Registers:**
SS:SP restored to value before INT B4h

**Conflicts:** Netroom NETSWAP4 (Chapter 41), ROM BASIC (Chapter 54)

**See also:** INT 2Fh Function C9FFh, INT B4h

## Trusted Access

Trusted Access is a security and access-control package by Lassen Software, Inc.; NB.SYS is a device driver to prevent the user from terminating CONFIG.SYS or AUTOEXEC.BAT with Ctrl-Break.

---

### INT 21h Function 3Fh

*Trusted Access*

### *NB.SYS - GET STATE*

**Registers at call:**
AH = 3Fh
BX = file handle for device "$$NB$$NB"
CX = 0002h (size of state)
DS:DX -> buffer for state record (see Table 66-12)

**Return Registers:**
CF clear if successful
    AX = number of bytes actually read (0 if at EOF before call)
CF set on error
    AX = error code (05h,06h) (see Table 24-2)

**See also:** Function 40h, Data Guardian Function 4101h (below)

**66-12 Format of state record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | 00h off, 01h on |
| 01h | BYTE | keys being disabled<br>bit 0: Ctrl-Break<br>bit 1: SysRq<br>bit 2: Ctrl and Alt<br>bit 3: Ctrl-Alt-Del<br>bit 7: all keys (overrides other bits) |

---

### INT 21h Function 40h
### *NB.SYS - SET STATE*

*Trusted Access*
*(undocumented)*

**Registers at call:**
AH = 40h
BX = handle for character device "$$NB$$NB"
DS:DX -> state record (see Table 66-12)
CX ignored

**Return Registers:**
CF clear if successful
  AX = number of bytes actually written
CF set on error
  AX = error code (05h,06h) (see Table 24-2)

**See also:** Function 3Fh

## VEDIT VSWAP

VSWAP is the resident portion of VEDIT's "swapper" capability; VEDIT is a programmer's text editor by Greenview Data. As with all programs using multiplex numbers in the range C0h to FFh, there are numerous potential conflicts (listed in Table 2-2); additionally, VSWAP conflicts directly with HEART.COM (Chapter 57).

---

### INT 2Fh Function D600h
### *INSTALLATION CHECK*

*VEDIT VSWAP*
*(undocumented)*

**Registers at call:**
AX = D600h

**Return Registers:**
AL = D6h if installed

**See also:** Functions D601h and D602h

---

### INT 2Fh Function D601h
### *UNDETERMINED FUNCTION*

*VEDIT VSWAP*
*(undocumented)*

**Registers at call:**
AX = D601h
BL = *subfunction number*

**Return Registers:**
BL = return code *unknown*
ES = resident portion's *data* segment
DX = resident portion's code segment

**See also:** Function D600h

---

### INT 2Fh Function D602h
### *EXEC PROGRAM WITH SWAP*

*VEDIT VSWAP*
*(undocumented)*

**Registers at call:**
AX = D602h
other registers set as for INT 21h
  Function 4B00h (Chapter 24)

**Return Registers:**
CF set on error
  AL = error code
    82h = failure due to *unknown*
CF clear on success

**See also:** Function D600h, MS-DOS INT 21h Function 4Bh (Chapter 24)

## WHOA!

WHOA!.COM is a system slow-down utility by Brad D Crandall.

---

### INT 2Fh Function 8900h
### *INSTALLATION CHECK*

*WHOA!.COM*

**Registers at call:**
AX = 8900h

**Return Registers:**
AL = status
  00h not installed
  FFh installed

**Conflicts:** none, but see Table 2-2
**See also:** Functions 8901h and 8902h

---

**INT 2Fh Function 8901h** *WHOA!.COM*

***UNINSTALL***

**Registers at call:**
AX = 8901h

**Return Registers:**
AL = status (FDh successful, FEh error)

**Conflicts:** none, but see Table 2-2
**See also:** Function 8900h

---

**INT 2Fh Function 8902h** *WHOA!.COM*

***SET DELAY COUNT***

**Registers at call:**
AX = 8902h
BX = delay count (larger values slow system down more)

**Return Registers:**
AL = status (FDh successful, FEh error)

**Conflicts:** none, but see Table 2-2
**See also:** Function 8900h

## XVIEW

XVIEW is a hypertext viewer by Flambeaux Software, Inc. The XVIEW API conflicts with GRIDLOC (below) and with Web (NI); in addition, Table 2-2 lists numerous potential conflicts with other software using multiplex numbers in the range C0h to FFh.

---

**INT 2Fh Function EE00h** *XVIEW*

***INSTALLATION CHECK***

**Registers at call:**
AX = EE00h

**Return Registers:**
AX = 00FFh if installed

---

**INT 2Fh Function EE01h** *XVIEW*

***POP UP GIVING TOPIC SEARCH KEYWORD***

**Registers at call:**
AX = EE01h
DS:DX -> ASCIZ string containing case-insensitive keyword to look up

**Return Registers:**
AX = status (00h,F1h,F2h) (see Table 66-13)

**Details:** The specified keyword should be a hyperlink in the _IndexPage of some database; the current database is searched first.
**See also:** Functions EE00h, EE02h, EE03h, EE04h, and EE06h

**66-13 Values for status:**

| | |
|---|---|
| 0000h | successful |
| 0001h | specified filename is not an xText database |
| 0002h | no databases found |
| 0003h | bad data in file |
| 0004h | memory shortage |
| 0005h | unable to open the requested file |
| 0007h | invalid page number for file |
| 00F1h | unknown subfunction |
| 00F2h | unable to pop up |

---

**INT 2Fh Function EE02h** *XVIEW*

***POP UP GIVING A PAGE NUMBER***

**Registers at call:**
AX = EE02h
DX = physical page number or anchor page number (see Table 66-14)

**Return Registers:**
AX = status (see Table 66-13)

**Details:** Physical page numbers are assigned by the hypertext compiler, and will change if a page is inserted in the middle.
**See also:** Functions EE00h, EE01h, EE05h, and EE06h

### 66-14 Values for anchor page number:

| | |
|---|---|
| FFEAh | _Credits |
| FFECh | _SearchTopics |
| FFEDh | _SearchText |
| FFF0h | _ManualList |
| FFF5h | _HelpOnHelp |
| FFF8h | _HomePage |
| FFF9h | _IndexPage |

## INT 2Fh Function EE03h

XVIEW

### POP UP GIVING FILENAME AND SEARCH TOPIC OR PAGE NUMBER

| Registers at call: | Return Registers: |
|---|---|
| AX = EE03h<br>DS:DX -> data packet (see Table 66-15) | AX = status (see Table 66-13) |

**See also:** Functions EE00h, EE01h, EE02h, and EE06h

### 66-15 Format of data packet:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | DWORD | -> ASCIZ database filespec (0000h:0000h for current database) |
| 04h | DWORD | -> ASCIZ text to look up or 0000h:0000h |
| 08h | WORD | page number (0000h if keyword used) |
| 0Ah | 6 BYTEs | reserved |

## INT 2Fh Function EE04h

XVIEW

### POP UP AND READ SCREEN FOR SEARCH TOPIC KEYWORD

| Registers at call: | Return Registers: |
|---|---|
| AX = EE04h | AX = status (see Table 66-13) |

**Details:** This call is equivalent to the action taken when the user presses the Alt-L hotkey.
**See also:** Functions EE00h, EE01h, EE03h, and EE06h

## INT 2Fh Function EE05h

XVIEW

### POP UP TO MOST-RECENTLY VIEWED PAGE

| Registers at call: | Return Registers: |
|---|---|
| AX = EE05h | AX = status (see Table 66-13) |

**Details:** This call is equivalent to the action taken when the user presses the Alt-H hotkey.
**See also:** Functions EE00h, EE02h, and EE06h

## INT 2Fh Function EE06h

XVIEW

### WAIT FOR POP-DOWN AND GET EXIT CODE

| Registers at call: | Return Registers: |
|---|---|
| AX = EE06h | AX = status (see Table 66-13) |

**Details:** Although this call is not required, the exit code can alert the caller to problems; if the call is not made, the program should enforce a delay of about 1/2 second to allow the viewer to pop up, and should not get keyboard input or attempt disk accesses during the delay.
**See also:** Functions EE00h to EE05h

## Miscellaneous

In a chapter totally dedicated to describing "those functions which do not neatly fit into any of the other chapters", this section holds the messiest misfits of them all. That's not to say that the programs discussed in this section are "misfits" but simply that their use of interrupts did not fit into the classification technique we chose in our efforts to bring some semblance of order into the chaos of reality. If you can't find what you're looking for elsewhere in this book, it's probably in this section.

---

### INT 16h Function 6A6Bh
*FastJuice (undocumented)*

### *DISABLE/UNLOAD*

**Program:** FastJuice is a resident battery-power monitor by SeaSide Software.

**Registers at call:**
AX = 6A6Bh

**Return Registers:**
*unknown*

**See also:** Function 7463h

---

### INT 16h Function 7463h
*FastJuice (undocumented)*

### *INSTALLATION CHECK*

**Registers at call:**
AX = 7463h ("tc")

**Return Registers:**
AX = 5443h ("TC") if installed

**See also:** Function 6A6Bh

---

### INT 16h Function 8765h
*AT.COM version 8/26/87*

### *API*

**Program:** AT.COM is a resident scheduler by Bill Frolik.

**Registers at call:**
AX = 8765h
BX = 4321h
CX = *unknown* or FFFFh
if CX = FFFFh
    DX = number of event to remove or FFFFh

**Return Registers:**
ES:BX -> event record array

**66-16 Format of event record:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | BYTE | in-use flag (00h free, 01h in use, FFh end of array) |
| 01h | BYTE | day of date on which to trigger |
| 02h | BYTE | month of date on which to trigger |
| 03h | BYTE | trigger time, minute |
| 04h | BYTE | trigger time, hour |
| 05h | WORD | offset of command to be executed |

---

### INT 16h Function 99h
*SCOUT v5.4*

### *GET UNKNOWN VALUE*

**Program:** Scout is a memory-resident file manager by New-Ware.

**Registers at call:**
AH = 99h

**Return Registers:**
AX = ABCDh
BX:CX -> *unknown* (appears to be start of PSP for resident portion)

**See also:** Function 9Eh

---

### INT 16h Function 9Eh
*SCOUT v5.4*

### *INSTALLATION CHECK*

**Registers at call:**
AH = 9Eh

**Return Registers:**
AX = ABCDh if installed

**See also:** Function 99h

## INT 16h Function EDh

*Borland Turbo Lightning*

***API***

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = EDh | |
| BH = EDh | |
| BL = function | |
| 00h installation check | AX = 5205h |
| | CH = major version |
| | CL = minor version |
| 01h *unknown* | |
| 02h get resident CS | AX = code segment of resident portion |
| 03h get resident *unknown* | AX = offset of some buffer in resident code seg |

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = EDh | |
| BH = EDh | |
| BL = function | |
| 04h *unknown* | |
| 05h set *unknown* | AX = status (0000h OK, 0001h out of range) |
| AL = 0 to 0Ch | |
| 06h *unknown* | |
| 07h *unknown* | |
| 08h *unknown* | AX = 0, 1 or 2 |
| AL = *character* | |
| CX = *unknown* | |
| DX = *unknown* | |
| 09h *unknown* | |
| 0Ah *unknown* | AX = *unknown* |
| CX = *unknown* | |
| DX = *unknown* | |

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = EDh | |
| BH = EDh | |
| BL = function | |
| 0Bh *unknown* | AX = 0, 40h, 80h |
| DS:SI -> *unknown* | |
| 0Ch *unknown* | AH = 0 |
| DS:SI -> *unknown* | AL = *unknown* |
| 0Dh set *unknown* | |
| (sets an internal flag) | |
| 0Eh *unknown* | AX = 0, 1 or 2 |
| DS:SI -> *unknown* | |
| 0Fh *unknown* | |
| 10h *unknown* | |

**Details:** AX in general returns an error code from most functions.

### INT 21h Function 4101h
### *UNDETERMINED FUNCTION*

*SoftLogic Data Guardian*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = 4101h | AX = 0000h if installed |
| DX = FFFEh | |

**Details:** The resident code sets several internal variables on this call.
**Conflicts:** MS-DOS Function 41h (Chapter 24)
**See also:** Trusted Access Function 3Fh (above), PC Tools DATAMON INT 16h Function FFA3h Subfunction 0000h (Chapter 61)

---

### INT 2Dh Function 10h
### *GET MONITORING INFORMATION*

*RAMLIGHT v1.0*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AL = 10h | ES:BX -> array of fake device driver headers used in monitoring |
| AH = AMIS multiplex number (see Chapter 50) | CX = *number of drives being monitored* |

---

### INT 2Fh Function 18h
### *UNDETERMINED FUNCTION*

*MS-Manager*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 18h | *unknown* |
| *others, if any, unknown* | |

---

### INT 2Fh Functions 47h, 4Dh, and AFh
### *UNDETERMINED FUNCTIONS*

*Unknown*
*(undocumented)*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 47h, 4Dh, and AFh | *unknown* |
| *others, if any, unknown* | |

**Details:** Function 47h is reportedly called by Microsoft BASIC Compiler v7.0, while Function 4Dh is reportedly called by the Kana Kanji Converter and MSKK.

---

### INT 2Fh Function F700h
### *INSTALLATION CHECK*

*AUTOPARK.COM*

**Program:** AUTOPARK.COM is a resident hard disk parker by Alan D. Jones.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = F700h | AL = status (00h not installed, FFh installed) |

**Conflicts:** none, but see Table 2-2

---

### INT 2Fh Function F701h
### *SET PARKING DELAY*

*AUTOPARK.COM*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AX = F701h | none |
| BX:CX = 32-bit count of 55ms timer ticks | |

**Conflicts:** none, but see Table 2-2

---

### INT 60h
### *USER INTERFACE FUNCTIONS*

*Atari Portfolio*

**Program:** This interrupt supplies a number of subfunctions which perform such functions as drawing boxes and menus, and provide input line editing.
**Conflicts:** see Table 2-3
**See also:** Atari INT 61h

## INT 60h

*PC-IPC*

***API***

**Program:** PC-IPC is a shareware TSR by Donnelly Software Engineering which allows communication between independent programs.

**Registers at call:**

STACK DWORD pointer to parameter block (see Table 66-17)

**Return Registers:**

STACK unchanged

**Details:** INT 60h is the default, any interrupt vector may be used by specifying the vector on the commandline.

**Conflicts:** see Table 2-3

### 66-17 Format of parameter block:

| *Offset* | *Size* | *Description* |
|---|---|---|
| 00h | WORD | caller's ID |
| 02h | WORD | to ID |
| 04h | WORD | command code (see below) |
| 06h | WORD | returned status (see below) |
| 08h | WORD | returned error code (see below) |
| 0Ah | WORD | size of data |
| 0Ch | DWORD | pointer to data buffer |

### 66-18 Values for command code:

| | | |
|---|---|---|
| 01h | "IPC_CMND_INQUIRE" | inquire current status<br>set status field, writes WORD to data buffer containing free message space in bytes, and sets the "size" field to the number of messages waiting |
| 02h | "IPC_CMND_ENABLE" | reenable PC-IPC<br>ignored unless called with the same ID that disabled PC-IPC |
| 03h | "IPC_CMND_DISABLE" | disable PC-IPC |
| 04h | "IPC_CMND_INSTALL" | reset PC-IPC |
| 06h | "IPC_CMND_RDATA" | read data<br>returns first message in data buffer, sets "size" to message length and "to ID" field to sender's ID<br>if no messages available, bit 4 of status is cleared and "size" is set to zero |
| 07h | "IPC_CMND_SDATA" | send data |
| 08h | "IPC_CMND_REQID" | require user ID<br>create a new recognized ID and return in "caller's ID" field |
| 09h | "IPC_CMND_DELID" | cancel user ID<br>delete caller's ID from pool of recognized IDs |
| 0Ah | "IPC_CMND_RDATAW" | read data, wait if no messages available |
| 0Bh | "IPC_CMND_VERS" | get PC-IPC version<br>string representing version returned in data buffer, "size" field set to length of string |

### 66-19 Bitfields for returned status:

| | |
|---|---|
| bit 0 | unused |
| bit 1 | IPC enabled |
| bit 2 | IPC installed |

| | |
|---|---|
| bit 3 | error |
| bit 4 | message(s) available |

**66-20 Values for error code:**

| | |
|---|---|
| 00h | no error |
| 01h | invalid command or parameter |
| 02h | only process 0 can install/reset IPC |
| 03h | process can not install/reset IPC |
| 04h | IPC is not enabled |
| 05h | process can not disable IPC |
| 06h | invalid destination process ID |
| 07h | invalid sending process ID |
| 08h | invalid data destination |
| 09h | no more process IDs available |
| 0Ah | can not relinquish that process ID |
| 0Bh | message space is full |
| 0Ch | IPC is not installed |

### INT 60h
***UNDETERMINED FUNCTION***

*PC/370 v4.2 (undocumented)*

**Program:** PC/370 is an IBM 370 emulator by Donald S. Higgins.

**Registers at call:**
*unknown*

**Return Registers:**
*unknown*

**Details:** This is the default interrupt, however the documentation includes instructions for patching the system for another interrupt.

**Conflicts:** see Table 2-3

**See also:** INT 2Fh Function 7F24h, PC/370 INT DCh

### INT 63h
***API***

*Kofax KF9X00 image manipulation card interface*

### INT 64h
***MicroECLIPSE COPROCESSOR INTERFACE***

*Data General DG10*

**Conflicts:** see Table 2-3

**See also:** DG10 INT 65h, DG10 INT 66h

### INT 64h
***API***

*Extended Batch Language v3.14+*

**Program:** Extended Batch Language is a batch-file enhancer by Seaware.

**Registers at call:**
AH = function
00h to 5Fh chained to previous handler
60h to 6Ch reserved, return immediately
80h to FFh chained to previous handler
6Dh (v4.01+) insert tone in queue
AL = *unknown*
CX = frequency in Hertz
DL = duration in clock ticks

**Return Registers:**
AL = status
00h if note stored
01h if no room to store

**Registers at call:**
AH = function
6Eh clear *unknown* counter/flag
6Fh return counter/flag that

**Return Registers:**

Subfunction 6Eh clears
70h *unknown*
AL = *unknown*
71h *unknown*
AL = *unknown*
72h *unknown*
73h insert byte at end of keyboard buffer
AL = byte to insert

AL = status
00h if byte inserted
01h if no room to store

**Registers at call:**
AH = function
74h insert byte at front of keyboard buffer
AL = byte to insert

**Return Registers:**
AL = status (as for Function 73h)

75h *unknown*
76h get keyboard "stack" status
AL = 'K' if kbd read will read physical keyboard
'S' if it will read EBL internal keyboard buffer
AH = *unknown*
77h clear internal keyboard buffer
78h *unknown*
AL = *unknown*
79h *unknown*
7Ah *unknown*
AL = *unknown*
7Bh *unknown*
AL = *unknown*
7Ch *unknown*
AL = *unknown*

**Registers at call:**
AH = function
7Dh *unknown*
AL = *unknown*
7Eh clear buffer for *unknown*
7Fh installation check

**Return Registers:**
CX = version in BCD
DI = segment of *unknown*
BX = segment of *next program's PSP*

**Details:** The chaining does not check whether the interrupt had been hooked before, so if you try to chain when the previous vector was 0000h:0000h, you'll be in trouble.

Functions 72h and 7Ah-7Dh appear to be interfaces to the optional floating-point and extended function packages

**Conflicts:** see Table 2-3

---

## INT 65h

*Data General DG10*

### *MicroECLIPSE COPROCESSOR INTERFACE*

**Conflicts:** see Table 2-3
**See also:** DG10 INT 64h, DG10 INT 66h

---

### INT 65h

*SD.COM v6.2*

#### *INVOCATION COUNT (NOT A VECTOR!)*

**Purpose:** The unregistered version of SD62.COM uses the low byte of this vector to count the number of invocations, displaying a registration reminder each time after the 20th use.
**Conflicts:** see Table 2-3

---

### INT 66h

*Data General DG10*

#### *MicroECLIPSE COPROCESSOR INTERFACE*

**Conflicts:** see Table 2-3
**See also:** DG10 INT 64h

---

### INT 6Ah

*OPTHELP.COM*

**Program:** OPTHELP is an optionally-resident help system for SLR Systems's OPTASM assembler.
**Details:** OPTHELP may be configured to use any interrupt from 60h to 7Fh (default 6Ah).
**Conflicts:** DECnet DOS (NI), Super-TCP (NI)

---

### INT 6Bh Function 6Bh

*Tandy SCHOOLMATE PLUS*

#### *API*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 6Bh | *unknown* |
| AL = E0h to FFh | |
| *others, if any, unknown* | |

**Conflicts:** Virus (Chapter 59), DECnet DOS (NI), NASI/NACS (NI), NCSI (NI), TelAPI (NI)

---

### INT 7Ah Function 7Ah

*Canon IX-30F Image Scanner SI3.SYS*

#### *INTERFACE*

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = 7Ah | AX = status (0000h success, 000Eh undefined function code) |
| *others, if any, unknown* | |

**Details:** This interface is normally on INT 7Ah but can be changed via device driver command line arguments.
**Conflicts:** AutoCAD Device Interface (above), GO32 (Chapter 37), PRINDIR (Chapter 15), Novell NetWare (NI), Topware NOS (NI), X.PC Packet software (NI), IBM 3270 Workstation Program (NI)

---

### INT 7Ch

*IBM REXX88PC command language (undocumented)*

**Details:** Details were not available at the time of writing.
**Conflicts:** PRINDIR (Chapter 15), GO32 (Chapter 37), SK-UPPS (NI)

---

### INT 7Dh

*YTERM 1.4 (undocumented)*

#### *CLOCK SUPPORT*

**Conflicts:** HyperPAD below, SCSILink (Chapter 11), GO32 (Chapter 37)
**See also:** YTERM INT 7Eh

---

### INT 7Dh

*HyperPAD v2.2*

#### *API ACCESS*

**Details:** This vector is hooked but immediately passed on (without checking whether the previous handler was 0000h:0000h). The sole purpose of this vector is to provide the address of the data area described below.
**Conflicts:** YTERM above, SCSILink (Chapter 11), GO32 (Chapter 37)

**66-21 Format of HyperPAD data area:**

| *Offset* | *Size* | *Description* |
|---|---|---|
| -16h | DWORD | pointer to *unknown* FAR function |
| -12h | DWORD | pointer to callback setting function<br>[C calling conventions, (*callback)(int (_loadds far *)()) ] |
| -0Eh | 6 BYTEs | signature "BRC001" |
| -08h | DWORD | pointer to previous INT 7Dh handler |
| -04h | DWORD | pointer to *unknown* data |
| 00h | | HyperPAD INT 7Dh handler |

---

**INT 7Eh** *RESERVED FOR DIP, Ltd. ROM LIBRARY*

***API***

**Conflicts:** YTERM below, GO32 (Chapter 37), XLOAD (Chapter 41)

---

**INT 7Eh** *YTERM 1.4*

***UNDETERMINED FUNCTION*** *(undocumented)*

**Conflicts:** DIP Ltd. ROM Library above, GO32 (Chapter 37), XLOAD (Chapter 41)

**See also:** YTERM INT 7Dh, YTERM INT 7Fh

---

**INT 7Fh** *YTERM*

***UNDETERMINED FUNCTION*** *(undocumented)*

**Conflicts:** see Table 2-4

**See also:** YTERM INT 7Eh

---

**INT 7Fh** *Canon IXHND2 Scanner Interface*

***API***

**Conflicts:** see Table 2-4

---

**INT 80h** *Q-PRO4*

***UNDETERMINED FUNCTION***

**Conflicts:** SoundBlaster SBFM (Chapter 22), BASIC (Chapter 54), PKTINT (NI)

---

**INT A4h** *Right Hand Man*

***API*** *(undocumented)*

**Program:** Right Hand Man is a TSR desk-top utility originally by Red E Products which has evolved into Futurus Team.

| **Registers at call:** | **Return Registers:** |
|---|---|
| AH = function number (v3.3 supports functions 00h-52h) | CF set on error<br>CF clear if successful |

**Details:** This interrupt is only hooked while popped up.

**Conflicts:** ROM BASIC (Chapter 54)

**See also:** INT 2Fh Function A4E0h

---

**INT DCh** *PC/370 v4.1-*

***API***

**Conflicts:** Screen Thief above, APL*PLUS/PC (Chapter 53), ROM BASIC (Chapter 54)

**See also:** PC/370 INT 60h

# INDEX

.DVP file format, 857
.EXE file format, 534
.PIF file format, 857
*Undocumented DOS*, 554, 1056
1024-cylinder limit, 215
2XON;
  installation check, 332
  uninstall, 332
A20, 35, 775, 777
ABIOS;
  initialization, 33, 34
  system parameter table, 33
absolute disk write, 599
absolute disk read, 599
acceleration profile, 449, 450, 455, 458
access programs;
  MAGic, 488
access rights, 613, 614
AccessDOS, 485
  API, 485
  configuration, 485
Acorn BBC Master 512, 83
  *see also Acorn*, 83
Acorn;
  control block for OSGBPB call, 84, 85
  file attributes, 85
  file parameters, 84
  file/directory info, 85
  file parameters, 84
  filing system, 84
  input, 83-85
  miscellaneous functions, 86, 87
  opening files, 83
  OSARGS, 84,
  OSASCI, 86
  OSBGET, 84
  OSBYTE, 87
  OSFILE, 85
  OSFIND, 83
  OSGBPB, 83
  OSNEWL, 86
  OSRDCH, 85
  OSWORD, 86
  OSWRCH, 86
  output, 83, 84, 86
Ad Lib SOUND.COM, 465
  active voice, 467
  installation check, 465
  melodic, 466
  mode, 466
  percussive, 466
  signature block, 466
  state, 466
initialization, 466
ADAPT, 167
  data area, 167
  installation check, 167
Adaptec controllers;
  disk parameters, 217-18
Adaptec;
  AHA-154xA controllers, 243
  ASPI, 221
adapters;
  HP 95LX external card, 28
address line 20—see A20, 35
Addstor, 332
Adobe Systems Inc., 301
ADP-60 controller, 218
Advanced SCSI Programming Interface—see ASPI,
Advanced Graphics 1024, 175
Advanced Power Management Specification—see APM,
Advantec, 1154
After Dark, 287
  blanking, 287-88
  blanking delay, 288
  hot key, 288
  installation check, 287
  new input, 287
  password, 289
  resident code, 287
  undetermined functions, 288-89
  VxD, 289
AI Architects;
  installation check, 735
AIA_OS device, 705
Alarm, 61-62
alarm;
  HP 95LX, 410-11, 417
aliases;
  ANARKEY, 276
AMIS;
  chained interrupts, 973
entry point, 972
  error codes, 973
  hotkey list, 974
  hotkeys, 974
  installation check, 972
  popping up, 973
  RATSR, 1064
  signature string, 972
  uninstall, 973
Amstrad PC1512;
  mouse, 463
Amstrad;
  CMOS memory, 87
  non-volatile memory, 87-88
  PC1512, 87
  ROS version check, 89
  video, 88-89
ANARKEY;
  aliases, 276
  ANARKMD, 276
  data structure, 275
  history buffer, 276
  installation check, 275
  suspending, 276
  undocumented functions, 275-77
ANARKMD, 276
ANSI sequences, 170
ANSI standards;
  X3.131 (SCSI), 221
ANSI drivers, 177
ANSI.SYS, 177, 629
  display information, 629
  installation check, 629
  miscellaneous requests, 629
  parameter block, 630
ANSIPLUS.SYS, 177
anti-viral tools, 1067, 1074
  F-DRIVER.SYS, 1075
  F-NET, 1075
  F-PROT, 1076
  FLU_SHOT+, 1076
  IBMAVSH, 1076
  ScanBoot, 1074
  TBSCANX, 1078
  ThunderByte, 1078
  VDEFEND, 1076
  VSAFE, 1074
  VSAFE, 1074
  VWATCH, 1074
APCAL, 1145, 1146
APL, 1013
  available memory, 1015
  beep, 1016
  clearing screen, 1016
  garbage collection, 1015
  graphics interface, 1016
  keystroke buffer, 1017
  names, 1015-16
  object creation, 1013-14
  print screen, 1016
  printer driver, 1016
  program termination, 1013
  R= space, 1016
  shape model, 1013
  timers, 1017
  variable outswapping, 1015-16
  virtual memory, 1014-15
APM, 371
  CPU busy, 373
  CPU idle, 373
  device ID, 374
  disabling power management, 374
  event broadcasting, 376
  event codes, 376
  initialization, 372
  installation check, 371
  polling frequency, 378
  power management events, 375
  power state, 373
  power status, 375
  power-on defaults, 375
  shutdown, 373
  stand-by mode, 374
  suspend system, 374, 376
  undocumented functions, 376
  version check, 371, 376
APPEND path, 631
APPEND;
  found name state, 631

  installation check, 630
  search path, 630, 631
  state, 631
  version check, 630, 631
AQUEDUCT, 273
area codes, 1082-84
ASAP, 485
  installation check, 485
  undetermined function, 485
ASCII table, 1146
ASCII, 1146
  installation check, 1146
ASCIZ string length, 591, 592
ASPI, 221
  CCB, 223
  SCSI Request Block, 221
ASPITAPE.SYS, 223
  device state, 224
  mtget parameter block, 224
  mtop parameter block, 224
ASSIGN, 632
  drive assignment table, 632
  installation check, 632
ASSOC;
  association test, 279
  installation check, 279
  version check, 279
AT.COM, 1170
AT&T 6300, 89, 90, 1140
AT&T VDC600, 139, 140
Atari Portfolio, 1172
  extended BIOS, 89
  user interface, 89
ATI Wonder;
  data area, 166
  VCONFIG, 166
  SMS.COM, 166
ATI video, 142
ATI VGA Wonder, 143
  BIOS entry point, 201
ATI;
  feature flags, 100
ATTRIB, 278
attribute mode control register, 123
AutoBraille, 486-88
AutoCAD, 1146-48
AUTOPARK.COM, 1172
autoswitch mode, 150
AUX device, 922, 925-27
AVAILDEV, 515
AVATAR Serial Dispatcher;
  initialization, 266
  shutdown, 266
AVATAR.SYS, 983
  data segment, 178
  installation check, 177
  keyboard input, 285
  setup, 177-78
  state characters, 178
  undocumented functions, 178-79
AX PC;
  country codes, 79
  error codes, 77
  keyboard, 78-79
  printer, 79
  screen memory address, 78
  shift keys, 79
  video, 77-78
  video scrolling, 78
Back & Forth, 929
  hotkey, 929
  installation check, 929
  version check, 929
BACK&FORTH;
  task information, 930
background color, 121, 124
background processing, 305, 313
  OS/2 compatibility box, 627
background execution, 603, 816
BACKTALK, 1085
BallPoint;
  button mask, 450
  information, 450
  rotation angle, 450
  see also mouse, 450
bank-switched memory, 737
BASIC, 1029, 1030
  ROM location, 35
  start Cassette BASIC, 58
BASICA, 1030
battery status, 375
baud rate, 926
BDOS calls, 619
beep, 1016
Bernd Schemmer, 1042
Bernoulli drives, 359
Better Software Technology, Inc., 966
Biologic, 804, 1144
BIOS model codes, 44
BIOS data area;
  extended, 46
BIOS configuration information, 42
BIOS Window Extension;
  attributes, 165
  window coordinates, 164
  window coordinates, 164
BIOS Parameter Block, 560
BIOS;
  C&T extensions, 144
bit 3 (character) device is CLOCK$ (all
    reads/writes
BitBLT, 909
BitFax Scheduler, 1109, 1110
  commblock, 1110
  installation check, 1109
  scheduling, 1109
  status, 1109
  transmitting, 1110
  uninstall, 1110
BITFAX.TRA file, 1110
bitmaps;
  mouse cursor, 442, 444
BITNET, 4
Blank-It, 289
  blanking, 291
  disabling, 290
  enabling, 290
  hotkeys, 291
  installation check, 290
  MS Windows support, 290
  timeout, 290
blinking characters, 120, 123
BLOC Publishing Corp., 337
block devices, 523
  IOCTL, 527
  IOCTL capabilities, 530
BMB Compuscience Canada Utilities, 978
BNFHIGH, 929
BNFLOW, 929
BNU FOSSIL;
  see also FOSSIL, 254
  installation check, 254
Boca Research Super X VGA—see Tseng,
    197
boot sector, 58
boot device key, 53
boot disk, 52
  reference partition, 53
boot device key, 53
boot drive, 513
bootstrap loader, 58
border colors, 121, 123, 124
BorlInternational, 715
BorlDOS extenders;
  RTM, 725
BorlC++, 715
BorlDOS extenders;
  DPMILOAD, 715
  TKERNEL, 726
BorlC++ IDE;
  installation check, 1019
Borland;
  THELP, 983, 1144
  Turbo Debugger, 1052
  Turbo Lightning, 1171
BOUND instruction, 22
BPB, 560
break, 62, 927, 947
breakpoint interrupt, 22
breakpoints, 802
  Soft-ICE, 1050-1051
bright background, 123
Bromfield Software Products, 1143
Brown, Ralf, 4
Btrieve, 1026, 1029
  execute operation, 1020
  installation check, 1026
  parameter record, 1026
  process ID, 1020
Budget Software Company, 305
BUFFERS=, 550
Buffit, 1145
bugs;
  AMI 386 BIOS, 118
  AST Premier 386 BIOS, 118
  Boca Research Multi-EGA, 124
  BorlDPMILOAD, 515
  Compaq DOS, 566
  DESQview, 827-874
  DR-DOS, 513-600
  EXEPACK, 533
  IBM VGA Adapter, 131
  IBM VGA BIOS, 123
  MS-DOS, 521-600
  Realtek RTVGA, 145
  Trident 8900 BIOS, 133
  Tseng ET-4000 BIOS, 198
  video fonts, 127
  video scrolling, 119-120
Burnout Plus 293-294
bus width;
  video, 195
Bustek BT-542, 243

CACHECLK;
  installation check, 361
CALCULATOR, 1140
callbacks;
  DPMI, 664-676
  IBMSND driver, 482
  Jetstream completion, 301
  QEMM-386, 756-760
  Software Carousel, 965
callouts;
  4DOS, 646
  AutoCAD Device Interface, 1148
  BorlC++ IDE, 1019
  COMMAND.COM, 633
  control-break, 62
  device busy, 41

  - device close, 38
  - device open, 38
  - device post, 42
  - diskless boot, 58
  - DOS kernel, 637-638
  - error message translation, 633
  - HIMEM.SYS, 798
  - keyboard intercept, 37
  - MDEBUG, 1042-1044
  - MS Windows, 806-816
  - NDOS, 647
  - OmniView, 945-946
  - OS/2 compatibility box, 627
  - PRINT, 313
  - program termination, 38
  - ROM BIOS tracing, 47
  - SmartDrive, 353
  - StackMan installation, 1166
  - Super PC-Kwik, 358-359
  - SysReq activity, 39
  - Task Switcher, 935-939
  - to NLSFUNC, 637-638
- CAM Control Block, 225
- CAM—see Common Access Method, 224
- Canon IXHND2 Scanner Interface, 1177
- Canon IX-30F Image Scanner, 1176
- canonical filenames, 572
- CAPDOS, 1039
  - capture queue, 1039-1040
  - collection, 1039
  - installation check, 1039
- capitalization, 575, 576, 591
- CappaCom programs, 985
- CapsLock, 1003
- Carbon Copy Plus, 1059
  - CCHELP, 1059
  - disconnecting, 1059
  - phone numbers, 1059
- Cardinal Technologies—see Tseng, 197
- carrier detect, 927
- CAS, 1127
  - aborting events, 1129
  - Intel SatisFAXtion, 1135-1138
  - autoreceive, 1132
  - cover page, 1138
  - diagnostics, 1137
  - error codes, 1137
  - event types, 1129
  - events, 1130-1134
  - external data block, 1131
  - file manipulation, 1130-1137
  - file transfers, 1133
  - hardware status, 1134-1136
  - installation check, 1127
  - phonebook, 1117-1131
  - queue, 1129-1134
  - sending files, 1137
  - task submisstion, 1127
- cassette;
  - data transfer, 33
  - motor control, 32
  - status, 33
- CCB, 225
- CD-ROM redirector;
  - execute command, 244
  - reset, 245
  - uninstall, 245
- CDS, 556
- CED, 278
  - installable commands, 283
  - installation check, 283
- CEMM, 796
  - private API, 800
- Central Point Software, 343
- change line, 210
- character sets;
  - double-byte, 77
  - user-definable, 77
- character conversion;
  - AX printer, 80
  - Japanese, 79
  - printer, 80
- character output;
  - BIOS, 122
  - redirection, 166
  - to screen, 121, 122, 172, 173
- character input;
  - from screen, 120
- character generator, 126
- character attributes, 132
- character devices;
  - I/O status, 524
  - IOCTL, 525
  - IOCTL capabilities, 530
- charcter input;
  - from keyboard, 256
- checksums, 592
- Chips Technologies, 146
- Chulalongkorn University, 1148
- CiriSOFT TSR interface, 979
- Cirrus Logic;
  - available modes, 195
  - bus width, 195
  - controller type, 194
  - get VGA type, 194
  - memory size, 194
  - retrace frequency, 196
  - user options, 195
  - version check, 194
- clear screen, 193
- clipboard;
  - bitmaps, 818
  - clearing, 818
  - closing, 819
  - compacting, 819
  - formats, 818
  - HP 95LX, 406-408
  - metafile format, 818
  - opening, 817
  - retrieving data, 818
  - retrieving data, 819
  - storing data, 818
- Cloaked BIOS, 686
- Cloaking, 683
  - MS Windows, 686
  - XMS, 686
  - CACHECLK, 361
  - caller registers, 685
  - calling convention, 785
  - calling real mode, 687
  - error codes, 683
  - GDT selectors, 683
  - GDT selectors, 683
  - I/O port trapping, 687
  - IRQ mapping, 687
  - passalong address, 685
  - passalong chain, 683
  - port-trapping passalong, 688
  - protected-mode code, 784
  - protected-mode interrupt vectors, 684
  - real-mode calls, 683
  - save/restore state, 686
  - secondary stack, 687
  - segment manipulation, 684
  - TSS, 687
  - VDS calls, 687
- clock tick, 63
- clock ticks, 1009
- CLOCK $ device, 557
- closing file, 398
  - by machine, 639
  - by name, 638
  - by process, 639
  - SHARE, 639
  - via FCB, 500
- CLS, 193
- cluster size, 514
- CMC International—see TARGA.DEV, 230
- CMDSPY, 1056
- CMOS memory;
  - Amstrad, 87
  - HP Vectra, 95
  - memory map, 87
  - Amstrad, 87
- code page;
  - changing, 638
  - hardware, 634
  - keyboard, 636
  - number, 620
  - preparing, 620
- CodeRunneR, 488
- Codeview information, 537
- Codeview for Windows, 735
- collating table, 576
- color plane enable register, 123
- color select register, 123
- colors, 120
  - border, 121-124
  - color mix registers, 154
  - Compaq ADAPT.COM, 167
  - DAC registers, 126
  - palette, 121, 124, 130, 188
  - palette registers, 123, 165
  - palette, 160-161
- Columbia Data Products, 242
- Columbia PCs;
  - ROM debugger, 1055
- COM1, 28
- COM2, 27
- COM3, 28
- COM4, 27
- COM8, 27
- COMBI-disk;
  - block information, 363
  - cache control, 362
  - data packet, 361
  - flushing, 362
  - hit rate, 361
  - installation check, 361
  - options byte, 363
  - resizing, 362
- commpacket, 362
- COMM-DRV;
- COMM-DRV;
  - initialization, 271
  - port control block, 271
- commlines;
  - Acorn BBC Master 512, 87
- command-line editing, 283
- commrecall;
  - ANARKEY, 276
- comminterpreter, 643
- COMMAND.COM, 643
  - back door, 643
  - commandline, 643
  - commexecution, 643
  - commline buffer, 644
  - commname buffer, 644
  - error tables, 594
  - HMA, 644
  - installable commands, 644

  installable commands, 644
  parameter errors, 595
  ROM version, 644
  shareable portion, 644
commandline recall, 617
commandline, 644
Commercial Software Associates, 901
common memory, 824
Common Access Method;
  callback functions, 230
  callback functions, 230
  control block, 225
  event capabilities, 229
  flags, 229
  flags, 228
  flags, 228
  function code, 225
  installation check, 230
  SCSI capabilities, 229
  SCSI interface, 224
  sending CCB, 225
  status, 228
  XPT/SIM, 225
Communicating Applications Specification—see CAS,
Communications FOSSIL, 262
communications;
  COMn, 27
  COMn, 28
  infrared data link, 28
COMMUTE, 1059
  installation check, 1060
  resident code segment, 1060
Compaq SLT/286;
  power management, 94
  standby mode, 90
Compaq Systempro;
  multiple processors, 92
  multiple processors, 92
  multiple processors, 92
Compaq SLT/286;
Compaq SLT/286;
  modem configuration, 91
Compaq SLT/286, 154
Compaq Systempro, 154
Compaq AG1024.SYS;
  RGDI, 176
  request packet, 176
Compaq DOS;
  bugs, 566
Compaq Computer Corp., 799
Compaq Expanded Memory Manager—see CEMM, 799
Compaq;
  CPU speed, 93
  keyboard type, 93
  LCD, 90
  LCD, 90
  plasma display, 90
  screen blanking, 90
  UILIB, 1025
  VGADI41.EXE, 1148
  WANG_ER.COM, 216
compressed executables, 534
Compressed Volume File, 325
compression—see disk compression, file compression,
CompuServe, 4
ComSoft, 901
COMTROL Hostess;
  configuration, 1040
  control program memory, 1040
  extended addressing mode, 1040
  firmware debugging port, 1040
  Turbo Debugger, 1040
  invoke debugger, 1040
COMTROL Hostess i/ISA debugger, 1040
Comwave Microfax Specification, 1125
CON device, 547
Concurrent DOS;
  version check, 615
  installation check, 615
Concurrent CP/M—see DRMDOS, 917
CONFIG.SYS, 1166
control block;
  Acorn OSFILE call, 85
  OSWORD call, 86
  OSGBPB call, 84
control-break, 62
control-Break checking, 513
Convertible—see IBM Convertible, 36
coprocessor status, 677
coprocessor emulation, 678

coprocessor;
  error, 29
  exception, 30
  installed, 31
  instructions, 23
  not present, 22
  protection violation, 24
  protection violation, 23
  Weitek, 31
Cornerstone Technology PG1600— see also PGS1600.DEV,
Corona/Cordata BIOS;
  graphics bitmap, 94
country codes;
  AX PC, 77
  keyboard, 78
  printer, 79
  video, 77
country-specific information, 515, 576
  capitalization, 576
  capitalization, 575
COUNTRY.SYS file, 620
COURIERS, 266
  aborting output, 268
  break, 268
  character input, 267
  character output, 268
  configuration, 267
  flushing buffers, 267
  initialization, 267
  installation check, 266
  port activity, 267
  shutdown, 268
  speed, 268
  status, 268
Cove Software, 278, 293, 462
CP/M;
  MS-DOS, 504
CPSCHED, 1086
  installation check, 1086
  installation check, 1086
  uninstall, 1086
CPTASK, 960
  active tasks, 961
  installation check, 961
CPU type, 48
CPU-generated interrupts, 21
CPU speed;
  Compaq, 92
CPU idle, 812
CPU;
  multiple, 97
  multiple, 92
  multiple—*see also multiple processors,*
CR0 register, 650
CR3 register, 652
creating file, 397, 518, 566, 580, 1094, 1095
  via FCB, 503
critical sections;
  PRINT, 313
critical sections, 404
critical error handler, 597
  error codes, 598
  experror, 633
  installation check, 632
critical sections;
  MS Windows, 809
  TopView, 827
  MultiDOS Plus, 890
  DESQview, 868
  MultiDOS Plus, 890
  DESQview, 868
  DESQview, 866
  DoubleDOS, 941
Cross Products, Ltd., 240
CRT controller;
  virtualization, 776
  QEMM-386, 759
CRTC registers, 175
Cswitch, 930
  CPU idle, 930
  executing programs, 932
  loading programs, 931
  message queue, 931
  messages, 932
  program termination, 932
  semaphores, 930
  semaphores, 930
  status, 933
  suspending tasks, 931
  task swapping, 932
  task priority, 931
  Task Control Block, 933
CTask, 969
Ctrl-Break, 1166
Ctrl-Alt-Del, 1154
Ctrl-Alt-Del, 1154
CtrlAlt Associates, 174
CTS/RTS, 256
CTS, 927
CU Writer, 1148
CUBIT;
  installation check, 334
  state, 334
  uninstall, 334
CUCKOO, 1145
current directory, 400
current directory structure, 555
cursor emulation, 130
cursor shape, 171
cursor position, 171
  mouse, 441
  TopView, 835
  TopView, 854
cursor, 988
  position, 118
  position, 118
  shape, 118
  text mode, 117
cut paste, 915
CVF, 325
Cyrillic, 281
Data Guardian, 1172
Data General DG10, 1174-76
DataBiten, 1019

DATAMON, 1077, 1085, 1090, 1099, 1100
    installation check, 1099
    installation check, 1089
date, 509
    AT&T 6300, 89
    day counter, 62
    determining, 410
    for file, 402
    parsing, 416
    real-time clock, 61
    see also time, 61
    setting, 410
DATEFIX, 1140
DBLSPACE.BIN—see DoubleSpace, 324
DBLSPACE.EXE, 325
DBOS, 734
    functions, 734
    uninstall, 734
DCC, 133
DDT—see device descriptor table, 50
debuggers, 1039
    Columbia ROM debugger, 1055
    COMTROL Hostess i/ISA, 1040
    Soft-ICE, 1050
    Turbo Debugger, 1052
    Zenith ROM debugger, 1054
debugging registers, 756
debugging exception, 803
debugging tools;
    CAPDOS, 1039
    MDEBUG, 1042
    MS Windows debugging kernel, 1056
    HEART, 1055
    ISR, 1057
    386 debugging exception, 21
    386MAX, 772
    386MAX, 771
    breakpoint interrupt, 22
    DPMI, 675
    OS/2 DOSKRNL symbols, 624
    ROM BIOS tracing, 47
    single step interrupt, 21
    watchpoints, 675
default drive, 401, 402, 498, 503
defect tables, 235
delay;
    European MS-DOS, 606
    MX5 Extended FOSSIL, 264
    SND, 478
delayed writes, 343, 346, 349, 353
deleting files, 400, 502, 520, 1130, 1094
Dell;
    model byte, 46
dempaging, 673
descriptors, 658
    access rights, 684
    aliases, 706
    aliases, 694
    allocation, 655
    allocation, 658
    allocation, 654
    real-mode addresses, 655
    creation, 706
    segment base address, 656
    segment limit, 657
    segment base address, 703
    segment limit, 684
    segment limit, 704
    TKERNEL, 727
DESK.OVL file, 1088
DESKTOP, 1088-92, 1094, 1097, 1099
    beep, 1097
    color schemes, 1089
    colors, 1098
    delays, 1094
    editor, 1091
    file management, 1094
    file management, 1095
    hotkeys, 1098
    installation check, 1097
    messages, 1093
    mouse cursor, 1099
    original screen, 1093
    overlays, 1089
    overlays, 1088
    resident mode, 1093
    screen output, 1097
    screen output, 1094
    screen data, 1093
    uninstall, 1098
    uninstall, 1095
    version check, 1090
    window parameters, 1091
    window management, 1098
DESQ, 883
DESQview/X;
    DOS/4GX, 709
DESQview/X;
    QEMM-386, 766
DESQview-awareness, 823
DESQview/X, 823
    undetermined functions, 878
    starting applications, 877
    XDI, 882
    X server, 870
    version check, 878
    direct window, 872
DESQview, 856
    see also DESQview, DVDOS4GX, TopView, 874
    display name, 878
    LPD, 865
    object bits, 862
    Network Manager, 865
    process manager, 874
DESQVIEW.DVO file, 863
DESQview;
    .DVP file format, 857
    .DVP variable-length records, 859
    ADDINSTANCEDATA function, 870
    APILEVEL function, 865
    APPNUM function, 864
    ASSERTMAP function, 860
    ASSERTMAP function, 867
    ASSERTVIR function, 869
    attaching windows, 860
    BEGINC function, 827
    bugs, 873
    bugs, 870
    bugs, 874
    bugs, 874
    bugs, 863
    bugs, 866
    closing objects, 852
    common memory, 824
    common memory, 864
    common memory, 880
    CONNECT function, 860
    CPU idle, 824
    critical sections, 827
    critical sections, 827
    critical sections, 868
    critical sections, 868
    critical sections, 866
    CSTYLE function, 866
    DBGPOKE function, 865
    DESQview directory, 870
DESQVIEW.DVO file, 863
DISPATCHINT function, 869
DISPATCHINTAFTERDOS function, 871
DISPEROR function, 828
DOSUSER function, 869
DVP processing, 881
DVP processing, 882
DVXMS.DVR, 885
EMM.DVR, 874
enabling extensions, 866
ENDC function, 827
ENTER function, 868
EOF on object, 853
erasing objects, 852
error popup, 828
error handling, 867
error handling, 867
error recovery, 863
error codes, 863
External Device Interface—see XDI, 879
FINDMAIL function, 865
FREEBIT function, 825
GETBIT function, 825
GETBUF function, 829
GETCOMMON function, 868
GETCRIT function, 866
GETDVPATH function, 870
GETERROR function, 867
GETFLAGS message, 862
GETFOREGROUND function, 870
GETMEM function, 824
GETMEM function, 865
GETNAME message, 861
GETPRI message, 861
initialization, 882
input from objects, 832
installation check, 873
Instance Item Structure, 871
ISOBJ function, 825
JUSTIFY function, 866
keyboard objects, 850
keyboard focus, 881
keyboard mouse, 860
keyboard focus, 870
KMOUSE function, 860
LOCATE function, 826
locking mailboxes, 856
mailbox messages, 831
mailbox names, 865
mailboxes, 849
mailboxes, 830
mailboxes, 850
mailboxes, 862
mailboxes, 865
mailboxes, 861
mailboxes

mapping context, 860
mapping context, 867
memory allocation, 824
memory allocation, 824
memory allocation, 868
memory available, 864
memory allocation, 868
memory available, 864
memory available, 864
memory allocation, 860
memory allocation, 865
memory allocation, 865
message interface, 830
messages, 863
NEWPROC function, 857

  object cursor, 854
  object destruction, 831
  object creation, 830
  object length, 848
  object handles, 825
  object size, 848
  object status, 853
  object bits, 849
  object bits, 851
  object types, 867
  object list, 871
  object bits, 862
  object bits, 862
  object bits, 862
  ObjectQ, 851
  ObjectQ, 861
  ObjectQ, 861
  objects, 825
  OBJNEXT function, 871
  Open Window menu, 863
  opening objects, 851
  OSTACK function, 827
  output to window, 834
  output to window, 849
  panel files, 831
  panels, 833
  PAUSE function, 824
  PGMINT function, 828
  pointer objects, 854
  pointer objects, 850
  pointer objects, 855
  positioning window, 828
  POSTTASK function, 829
  POSWIN call, 828
  PRINTC function, 825
  priority, 861
  priority, 861
  process status, 872
  PROCESSMEM function, 870
  program name, 863
  proramming error popup, 826
  PUSHKEY function, 866
  PUTCOMMON function, 868
  PUTKEY function, 869
  PUTMEM function, 824
  PUTMEM function, 865
  QEMM hooks, 759
  QEMM hooks, 760
  QEXT.SYS, 883
  reading objects, 854
  READINTO message, 862
  refreshing display, 855
  screen contents, 832
  screen output, 850
  screen output, 825
  screen updates, 824
  SCRNINFO function, 869
  see also TopView, DESQview/X, 856
  SETBIT function, 825
  SETERROR function, 867
  SETFLAGS message, 862
  SETPRI message, 861
  shadow screen, 823
  shadow screen, 824
  shadow screen, 829
  SOUND function, 826
  START function, 827
  starting applications, 857
  STOP function, 827
  suspending tasks, 827
  suspending tasks, 827
  suspending tasks, 829
  swapping processes, 882
  swapping processes, 882
  Switch Windows menu, 864
  system memory block, 860
  system memory, 865
  system memory, 865
  system memory, 860
  task creation, 830
  task status, 872
  timers, 833
  true video parameters, 869
  undetermined functions, 864
  undetermined functions, 870
  undetermined functions, 868
  undetermined functions, 864
  undetermined functions, 864
  undetermined functions, 871
  USTACK function, 829
  version check, 873
  video, 856
  video, 857
  video, 856
  VIDEONOTIFY function, 872
  viewport, 866
  virtualized screen, 869
  window locations, 826
  window information, 871
  window information, 872
  WININFO function, 871
  writing to window, 834
  writing to objects, 833
device descriptor table;
  get table, 51
device busy, 41
device close, 38
device open, 38
device descriptor table;
  number of tables, 50
  indicator flags, 51
  find entry, 51
device post, 42
  find entry, 50
  format, 50
  find entry, 51
device driver;
  Stacker, 332
  Stacker, 331
  Stacker, 331
  Stacker, 331
  SmartDrive, 353
device types, 528
device information word, 522
  device chain, 593
  request header, 583
  special device, 601
  error codes, 585
device driver attributes, 557
device driver;
  commcodes, 582
  adding new devices, 582
device driver header, 556
device driver;
  European MS-DOS helper functions, 585
device driver lookahead flag, 574
device drivers;
  PC-MOS/386, 959
devices;
  control channel, 523
  control channel, 523
  control channel, 523
  device type codes, 42
  information, 523
  information, 522
  IOCTL, 523
  IOCTL, 523
  IOCTL, 523
  remote, 524
  removable, 524
DGIS;
  available devices, 166
  output device, 166
  redirection, 166
diagnostics;
  disk controller, 209
  disk drive, 209
  disk controller, 209
  NewSpace, 330
  power-on self-test, 34
  TARGA.DEV, 232
Diamond Speedstart;
  24-bit graphics modes, 201
  sync parameters, 201
DIET, 321
  control flags, 322
  control flags, 322
  decompression, 323
  decompression, 324
  EMS status, 323
  expprogram, 324
  external procedure, 323
  external procedure, 323
  external procedure, 323
  installation check, 324
  installation check, 321
  options, 322
  options, 322
  options, 322
  overlay mode, 324
  overlay mode, 324
  resident code segment, 321
  temporary directory, 322
  temporary directory, 322
  uninstall, 323
Digiboard, 250
  BIOS command, 253
  CCB command, 252
  CCB command, 252
  CH_KEY_RDY flag, 251
  flushing buffers, 252
  flushing buffers, 252
  flushing buffers, 251
  input availability, 251
  input availability, 251
  protocol, 250
  read string, 251
  special characters, 253
  special characters, 253
  speed, 250
  status check, 250
  string output, 251
  transmit buffer space, 252
digital sound;
  start/stop, 478
  installation check, 477
  start/stop, 478
Digital-to-Analog Converter, 125
Digital Research;
  single-tasking operating systems, see also Concurrent DOS, DR-DOS, DOS Plus, 619
  BDOS calls, 619
  multiuser operating systems, 616
  DOS Plus, 917
  DRMDOS, 917
  GEM, 1030
Digital Communications Associates, Inc., 1127
digitized sound;
  playing, 480

digitized sound;
  capabilities, 482
digitizer, 1147
  Houston Instruments HiPad, 1147
  status, 1147
DIP, Ltd. ROM Library, 1177
Direct Graphics Interface Standard— see DGIS,
Direct Access, 1029
directories;
  creating, 401
  creating, 517
  creating, 626
  current, 400
  current, 400
  current directory, 517
  removing, 401
  removing, 517
directory entry, 501
disk types, 210
disk parameters, 212
disk types, 210
disk parameter table, 211
disk compression;
  DoubleSpace, 324
  NewSpace, 329
disk caches;
  flushing, 324
  flushing, 324
  NOW!, 341
  PC-Cache, 343
  IBMCACHE.SYS, 360
  HyperDisk, 339
  read-ahead, 346
  QuickCache II, 344
  read-ahead, 349
  flushing, 362
  hit rate, 342
  hit rate, 343
  hit rate, 352
  hit rate, 361
  flushing, 357
  size, 341
  SmartDrive, 350
  status, 350
  status, 353
  size, 340
  size, 338
  resizing, 345
  resizing, 351
  resizing, 362
  sector locking, 348
  resizing, 337
  Super PC-Kwik, 354
  double buffering, 353
  enabling, 344
  enabling, 344
  enabling, 348
  disabling, 358
  disabling, 351
  delayed writes, 353
  disabling, 343
  disabling, 344
  disabling, 348
  enabling, 358
  FAST!, 337
  flushing, 344
  flushing, 346
  flushing, 350
  flushing, 351
  flushing, 344
  flushing, 344
  FAST!, 339
  flush interval, 345
  flushing, 343
  flushing, 343
  flushing, 352
  disabling, 337
  cache locking, 351
  audible feedback, 346
  delayed writes, 343
  cache size, 353
  CACHECLK, 361
  delayed writes, 349
  delayed writes, 346
  delayed writes, 337
  COMBI-disk, 361
  delayed writes, 343
  delayed writes, 349
disk space, 514
disk buffers, 498
disk file, 522
disk buffer flags, 559
disk buffer info, 558
disk buffer chain head, 558
disk buffers, 589
  unreferenced, 590
disk caches;
  NCACHE, 1103
  flushing buffers, 1103
Disk Manager, 215
Disk Spool II;
  spool file status, 309
  spool file, 309
  spool file, 309
  spooler control, 306
  spooler control, 307
  spool file, 306
  spool file, 306
  spool file, 307
  spool file, 308
  spool file, 306
  spooling existing files, 310
  spooling existing files, 310
  status, 308
  undetermined functions, 310
  spooler state, 308
  spooler state, 307
  spooler state, 306
  spooler state, 306
  spooler state, 306
  spooler control, 308
  spool file status, 309
  despooler control, 308
  despooler state, 306
  despooler state, 306
  despooler state, 306
  despooler control, 307
  despooler control, 306
  API, 306
  API, 307
  despooler state, 307
  despooler state, 308
  menu, 306
  menu, 307
  printer port, 309
  printer port, 310
  resident code segment, 306
  installation check, 307
  disabling, 308
  installation check, 306
  installation check, 306
  installation check, 306
Disk Transfer Area, 503
disk;
  3-mode floppy drive, 216
  4+Power floppy controller, 217
  absolute read/write, 599
  absolute read/write, 599
  ADP-60 controller, 218
  ADP-60 controller, 218
  available space, 327
  BIOS diskette handler, 211
  change line, 210
  configuration, 213
  configuration, 213
  diagnostics, 209
  diagnostics, 209
  diagnostics, 209
  formatting, 212
  formatting, 213
  formatting, 210
  formatting, 234
  formatting, 235
  formatting, 206
  formatting, 205
  formatting, 205
  formatting, 206
  media type, 210
  media type, 210
  parameters, 218
  parameters, 218
  parameters, 217
  parameters, 218
  parameters, 217
  parameters, 218
  parameters, 218
  parameters, 218
  parameters, 217
  parameters, 211
  parameters, 217
  parameters, 206
  parameters, 211
  parameters, 207
  parameters, 217
  parameters, 217
  parameters, 212
  parking heads, 211
  Pdisk, 218
  Pdisk, 218
  Pdisk, 218
  Pdisk, 217
  Priam controllers, 215
  Priam controllers, 214
  readiness, 209
  reading, 207
  reading, 208
  reading, 204
  recalibrating, 209
  reset, 208
  reset, 203
  resetting, 498
  Seagate controller, 214
  Seagate controller, 214
  Seagate controller, 214
  Seagate controller, 214
  sector buffer, 209
  sector buffer, 208
  seeking, 208
  status, 209
  status, 203
  status, 203
  TI Professional PC, 216
  type, 210
  verifying sectors, 205
  Western Digital Super BIOS, 214
  writing, 209
  writing, 208
  writing, 204
diskette parameters, 211
diskette parameter table, 211
diskless boot, 58

diskless workstations, 641
DISKMON.INI file, 1102
DISKMON;
  flushing buffers, 1103
DISKREET, 1102
display type, 132
display combination code, 133
display page;
  active, 119
display combination code, 132
display type, 190
display memory, 150
display drivers, 163
display pages;
  mouse, 446
  mouse, 446
DISPLAY.SYS, 633
  active code page, 633
  active code page, 633
  code page information, 634
  code page information, 634
  hardware code pages, 634
  installation check, 633
  installation check, 634
  undetermined functions, 634
  undetermined functions, 634
division by zero, 21
DJ Delorie, 733
DJGPP, 733
dLite;
  compressed file size, 335
  compressed files, 335
  compressed files, 335
  parameter block, 334
  parameter block, 334
  uncompressed file size, 335
DLL, 1021
DMA arbitration, 49
DMA, 365
DMA channels;
  MVSOUND.SYS, 472
DMA buffers;
  PCM.COM, 475
  PCM.COM, 474
  PCM.COM, 474
DMA;
  address translation, 370
  address translation, 369
  printer, 300
  EMS, 744
  VADMAD virtual device, 811
  buffer allocation, 368
  buffer allocation, 368
  buffer size, 767
  buffers, 369
  buffers, 369
  locking, 367
  locking, 367
  locking, 367
  locking, 368
DMQS, 137
Donald S. Higgins, 1174
Donnelly Software Engineering, 1173
DOS Parameter List, 567
DOS/V—see Far East DOS, 596
DOS/K—see Far East DOS, 596
DOS/J—see Far East DOS, 596
DOS Parameter List, 639
DOS Protected-Mode Interface—see DPMI, 653
DOS extenders;
  Phar Lap, 689
DOS Protected Mode Services—see DPMS, 679
  OS/386, 705
DOS/4G;
  protected-mode API, 710
  see also DVDOS4GX, 709
  undetermined functions, 710
  installation check, 710
  initialization, 710
DOS/16M;
  installation check, 709
  XBRX structure, 709
DOS/4G;
  undetermined functions, 710
DOS extenders;
  DPMILOAD, 715
  GO32, 733
  installation check, 736
  miscellaneous, 735
  DBOS, 734
  Borland, 715
  FlashTek, 689
  AI Architects, 735
  OS/286, 705
DOS/4G;
  undetermined functions, 711
  undetermined functions, 712
  undetermined functions, 712
  undetermined functions, 712
  undetermined functions, 713
  undetermined functions, 713
  uninstall, 710
  undetermined functions, 713
  undetermined functions, 712
  undetermined functions, 712
  undetermined functions, 711
  undetermined functions, 711
  undetermined functions, 711
  undetermined functions, 711
  undetermined functions, 711
  undetermined functions, 712
  undetermined functions, 711
DOS extenders;
  RTM, 725
DOS/4G;
  installation check, 713
DOS extenders;
  Windows/286, 735
  TKERNEL, 726
DOS/4G;
  undetermined functions, 713
DOS/4GW, 875
DOS/4GX;
  see also DOS/4G, DESQview/X, 874
DOS Plus;
  XIOS calls, 920
DOS Task Switcher—see Task Switcher, 935
DOS-UP.SYS;
  installation check, 1105
DOS—see MS-DOS, 493
DOSED, 283
DOSKEY, 277
  input line, 277
DOSKEY;
  installation check, 277
  line buffer, 277
DOSKRNL, 624
DOSMGR, 814
DOSMGR;
  patch table, 816
DOSSHELL, 933
  COMMAND.COM interface, 933
  COMMAND.COM interface, 934
  installation check, 933
  SHELLC interface, 933
  task switching API, 934
double fault, 22
double-byte character sets, 77
double buffering, 353
double-byte characters;
  lead byte table, 576
  lead byte table, 574
  lead byte table, 573
DoubleDisk—see DUBLDISK.SYS, 333
DoubleDOS, 939
DoubleDOS:virtual screen, 942
DoubleDOS;
  clearing typeahead, 940
  CPU idle, 942
  CPU idle, 943
  CPU idle, 942
  critical sections, 942
  critical sections, 941
  critical sections, 941
  critical sections, 942
  installation check, 940
  keyboard control flags, 940
  keyboard control flags, 941
  menu control, 939
  partitions, 941
  partitions, 940
  partitions, 940
  program status, 940
  program status, 940
  program priority, 941
  program priority, 941
  stuffing keystrokes, 940
  stuffing keystrokes, 940
  undetermined functions, 942
  virtual screen, 942
DoubleSpace, 324
  activation record, 326
  MRCI, 328
  Stealth, 767
  available space on drive, 327
  Compressed Volume File, 325
  device driver entry points, 326
  device driver entry points, 325
  DISK_UNIT structure, 326
  DISK_UNIT structure, 327
  drive mapping, 325
  drive swapping, 325
  drive mapping, 325
  DSPACKET structure, 324
  entry points, 328
  entry point record, 328
  error codes, 325
  flushing cache, 324
  flushing cache, 324
  fragment heap, 327
  free space record, 327
  installation check, 325
  mounting drives, 326
  mounting drives, 326
  relocating driver, 327
  relocating driver, 327
  sequence number, 325
Douglas Sisco, 483
DPB, 504
DPL, 567
DPMI, 653
  address mapping, 673
  address mapping, 673
  alias descriptors, 657
  Netroom, 785
  available memory, 667
  available memory, 671
  calling real mode, 664
  calling real mode, 664

  calling real mode, 663
  capabilities, 666
  capabilities, 667
  coprocessor emulation, 678
  coprocessor status, 677
  CPU idle, 809
  debug watchpoints, 675
  debug watchpoints, 675
  debug watchpoints, 675
  debug watchpoints, 675
  dempaging, 673
  descriptor allocation, 654
  descriptor allocation, 655
  descriptors, 658
  descriptors, 658
  descriptors, 659
  descriptors, 658
  descriptors, 657
  detect mode, 653
  discarding pages, 673
  discarding pages, 673
  error codes, 655
  host description, 666
  host description, 667
  IF virtualization, 674
  IF virtualization, 674
  IF virtualization, 674
  installation check, 653
  memory information, 671
  memory mapping, 670
  memory information, 667
  memory block information, 670
  memory allocation, 669
  memory mapping, 670
  memory-mapped devices, 670
  memory allocation, 667
  memory allocation, 668
  memory allocation, 668
  memory allocation, 668
  mode switch, 665
  MS-DOS memory allocation, 659
  MS-DOS memory allocation, 659
  MS-DOS memory allocation, 659
  page size, 672
  page locking, 671
  page locking, 671
  page attributes, 669
  page attributes, 669
  page attributes, 669
  paging, 656
  paging, 656
  paging candidates, 672
  paging, 672
  paging, 672
  processor exceptions, 662
  processor exceptions, 662
  processor exceptions, 662
  processor exceptions, 663
  processor exceptions, 660
  processor exceptions, 660
  protected-mode interrupt vectors, 661
  protected-mode interrupt vectors, 662
  QDPMI, 1108
  real-mode addresses, 655
  real mode callbacks, 665
  real-mode interrupt vectors, 660
  real-mode interrupt vectors, 660
  real mode call structure, 663
  real mode callbacks, 664
  real-mode memory, 672
  real-mode memory, 672
  resident service provider structure, 676
  resident handler callback, 676
  save/restore state, 665
  segment manipulation, 657
  segment manipulation, 657
  segment manipulation, 657
  segment manipulation, 656
  segment manipulation, 656
  selector locking, 656
  selector locking, 656
  selector increment, 656
  serialization, 677
  serialization, 677
  shared memory, 676
  shared memory allocation request structure, 676
  shared memory, 677
  shared memory, 677
  shared memory, 677
  terminate stay resident, 676
  vendor names, 654
  vendor-specific entry point, 654
  vendor-specific entry point, 674
  version check, 666
DPMILOAD, 715
DPMILOAD data, 717
DPMILOAD;
  calling real mode, 720
  descriptors, 720
  descriptors, 721
  error codes, 717
  executing programs, 717
  installation check, 715
  installation check, 721
  LDT, 721
  memory control record, 718
  memory block information, 716
  memory allocation, 715
  memory allocation, 716
  memory available, 716
  memory allocation, 719
  memory allocation, 719
  name list, 718
  processor exceptions, 723
  processor exceptions, 724
  protected-mode interrupt vectors, 724
  protected-mode interrupt vectors, 724
  segments, 721
  segments, 724
  selectors, 721
  selectors, 724
  undetermined functions, 719-25
DPMS, 679
  callup/down register structure, 682
  error codes, 682
  installation check, 679
dPressed files, 335
DR-DOS;
  bugs, 513
  passwords, 499
  passwords, 518
  passwords, 503
  passwords, 501
  bugs, 521
  kernel entry jump table, 618
  KEYB, 621
  KEYB, 621
  passwords, 614
  NLSFUNC, 619
  MEMMAX, 619
  installation check, 616
  HMA memory block, 618
  GRAFTABL, 620
  file owner, 615
  file attributes, 614
  GRAFTABL, 621
  HIDOS.SYS, 619
  history buffer, 617
  HILOAD, 617
  passwords, 614
DR MultiUser DOS;
  API, 619
DR-DOS;
  undetermined functions, 619
  version check, 616
  video memory space, 619
  TaskMAX, 618
  SHARE, 617
  see also Novell DOS, 613
  SHARE, 617
  file access rights, 614
  file access rights, 614
  bugs, 600
  bugs, 599
  code pages, 620
  country information, 619
  file access rights, 613
  passwords, 616
  see also Concurrent DOS, 613
  file owner, 615
  bugs, 570
  bugs, 578
  bugs, 562
  bugs, 546
  bugs, 545
DR Multiuser DOS—see DRMDOS, 917
DR-DOS;
  TaskMAX—see TaskMAX, 911
drive swapping, 325
drive data table, 588
Drive Parameter Block—see DPB, 504
DRIVEMAP, 1085
DRIVER.SYS, 530
DRIVER.SYS support;
  drive data table, 588
  device driver request, 582
  installation check, 582
  adding devices, 582
drives;
  default, 401
  default, 402
DRMDOS, 917
  AUX device, 922
  AUX device, 927
  AUX device, 926
  AUX device, 927
  AUX device control block, 927

  AUX device parameter block, 926
  AUX device, 926
  AUX device, 925
  AUX device, 924
  character control block, 926
  detaching from console, 924
  error codes, 921, 923
  global memory, 923
  message queues, 923-24
  process dispatcher, 924
  process priority, 924
  queue descriptor, 923
  queue parameter block, 923
  suspending tasks, 927
DSPACKET structure, 324
DSR, 927
DTA, 503, 510, 1165
DTR control, 255
DUBLDISK, 883
DUBLDISK.SYS, 527
  data record, 333
  installation check, 333
  signature data table, 333

DUP, 531
DUP2, 531
DVDOS4GX.DVR, 709
DVDOS4GX;
  installation check, 874
  LOAD32.EXP, 875
  mailboxes, 877
  mailboxes, 877
  mailboxes, 877
  process manager, 874
  program loader, 875
  resident code segment, 876
  resident code segment, 876
  task handle, 876
  undetermined functions, 874-77
DVeop, 883
DVSI, 884
DVSIXDI, 884
  process information, 885
  process information, 885
DVTMAN, 884
DVTree, 884
DVTXDI, 884
DVXMS.DVR, 885
dynamic link library manager, 1021

Eagle PC2, 141
EBIOS;
  break, 269
  configuration, 269
  input, 269
  input, 269
  installation check, 268
  recapturing port, 269
  status, 269
EEMS;
  accelerator card support, 745
  memory mapping, 746
  page frame addresses, 745
  page mapping, 746
  physical window array, 745
  see also EMS, 737
EFMT.EXE, 215
EGA registers;
  reading, 155
  installation check, 817
EGA.SYS;
  version check, 817
EGA;
  ROM BIOS, 138
  fonts, 138
  get information, 129
  Register Interface Library, 155
  vertical retrace, 27
  video modes, 138
  virtualization, 760
EGAPAL.COM;
  installation check, 174
EISA, 65
  32-bit calls, 67
  erase CMOS, 66
  error codes, 66
  expansion card ID, 67
  expansion card ID, 65
  get info, 66
  get info, 65
  get info, 67
  installation check, 65
  slot types, 65
  slots, 66
  update CMOS, 66
ELRES, 1142
Embedded DOS, 627
EMM Import structure, 764
EMM386, 365
  entry point, 799
EMM386.EXE;
  installation check, 799
  memory manager information, 797
  memory manager information, 797
  resident code segment, 797
  Digital Research, 619
EMMXXXX0 device, 769
EMS, 737
  alternate map register sets, 744
  alternate mapping sets, 803
  alternate map registers, 802
  TaskMAX, 911
  copy data, 743
  copying memory, 743
  DMA register sets, 744
  EEMS, 745
  error codes, 737
  handle directory, 742
  handle information, 740
  handle directory, 742
  handle name, 741
  handle count, 788
  handle attributes, 741
  handles, 739, 802
  hardware configuration array, 744
  hardware information, 743
  hidden page frames, 641
  I/O ports, 739
  manager status, 737
  mappable address array, 743
  mapping context, 739
  mapping context, 739
  mapping pages, 741
  mapping array, 741
  mapping registers, 740
  mapping context, 1107
  memory allocation, 738, 739
  memory mapping, 739
  number of pages, 738
  OS functions, 745
  page map alteration, 742
  page allocation, 744
  page map alteration, 742
  page map, 740
  page map, 740
  page frame segment, 738
  raw pages, 744
  translation array, 739
  version check, 739
  warm boot, 745
Emulaser ELTSR;
  undetermined functions, 303-4
  ScrollLock state, 304
Emulaser TSR;
  deactivating, 303
Emulaser ELTSR;
  uninstall, 303
  background processing, 305
  capture file, 303
  printing, 304
Emulaser ELSPL, 310
Emulaser ELTSR;
  capture file, 304
  activity indicator, 304
  install handlers, 302
  capturing output, 302
  capture file, 304
enhanced keyboard, 56
entry points;
  DR-DOS, 618
  error retrieval, 595
  FASTOPEN, 593
  QEMM VCPI, 649
  VCPI, 649
  DPMI, 654
  DPMI, 674
  Phar Lap DOS extender, 702
  OS/x86, 705
  MS Windows, 810
  MS Windows, 805
  XMS v2+, 747
  XMS v1.x, 747
  HIGHUMM.SYS, 800
  memory managers, 795
  memory manager
  EMM386.EXE, 799
  QEMM-386, 753
  RM386, 784
  QEMM-386, 756
  QEMM-386, 753
  XMS, 784
  QEMM-386, 754
  TKERNEL, 726
  QEXT.SYS, 883
  Task Switcher, 937
  TurboPower TSRs, 978
  AMIS, 972
  Alternate Multiplex Interrupt, 972
  OmniView, 945
  VIDRAM, 1107
  VGADI41, 1148
  AutoCAD video driver, 1148
  pcANYWHERE IV/LAN, 1063
  Buffit, 1145
environment, 507
environment variables;
  VER, 615
EP.INI file, 1102
equipment list, 31
Ergo;
  installation check, 707
Eric Bohlman, 490
Eric Isaacson, 1081
error detection;
  PCMCIA Socket Services, 76
  PCMCIA Socket Services, 76
error codes;
  cassette, 33
  device descriptor table, 50
error detection;
  PCMCIA Socket Services, 76
error codes;
  memory cache control, 48
error detection;
  PCMCIA Socket Services, 76
error codes;
  EISA System ROM, 66
error detection;
  PCMCIA Socket Services, 75
  PCMCIA Socket Services, 75
error codes;
  Video FOSSIL, 170
  TARGA.DEV, 230
  BIOS disk functions, 203
  PRINT, 314
  PC Paint Plus, 318
  DoubleSpace, 325
  VDS, 366
  HP 95LX, 388
  HP 95LX, 419
  pointing device, 437
  HP 95LX, 406
  MS-DOS, 493, 505
  MAGic, 488
  absolute disk read/write, 599
error tables, 594

error information;
    European MS-DOS, 603
error codes;
    critical error, 598
error tables, 594
error codes;
    device drivers, 585
    DPMI v1.0, 655
    Phar Lap 386/DOS-Extender, 691
    FlashTek X-32VM DOS extender, 691
    Phar Lap VMM, 695
    DPMS, 682
    Cloaking, 683
    RM386, 790
    EMS, 737
    DPMILOAD, 717
    XMS, 751
    MEMLIMIT, 782
    MultiDOS Plus, 887
    DRMDOS, 923
    MultiDOS Plus, 888
    MultiDOS Plus, 897
    Concurrent CP/M, 921
    DESQview, 863
    DRMDOS, 921
    Software Carousel, 963
    AMIS, 973
    PC-MOS/386, 958
    FGDRIVER, 997
    Image Processing Interface, 1032
    pcANYWHERE, 1064
    Soft-ICE, 1051
    PC-IPC, 1174
    Turbo Debugger, 1052
    XVIEW, 1168
    MDEBUG, 1049
    MDEBUG, 1045
errors;
    parameter errors, 595
ESaSS B.V., 1165
ESDI controllers, 212
ESDI;
    address translation, 213
    configuration, 213
    configuration, 213
    format modifiers, 212
    formatting, 213
    formatting, 212
    manufacturing header, 212
    periodic interrupt, 213
    POS information, 213
European MS-DOS, 585, 603-11
Evans & Sutherland, 301
event wait, 39
Everex Viewpoint VGA, 138
Everex Extended Video BIOS, 151
Everex Micro Enhancer EGA, 138
Exact Automatisering B.V., 1030
Exact, 1030
exception handlers, 803
Exception #13, 24
exceptions, 21
    alignment check, 24
    bounds-check, 22
    breakpoint, 22
    coprocessor, 23
    debug, 21
    divide error, 21
    double, 22
    DPMI exception handlers, 662
    general protection violation, 24
    invalid opcode, 22
    invalid task state segment, 23
    no coprocessor, 22
    overflow, 22
    page fault, 24
    segment not present, 23
    single-step, 21
    stack fault, 24
EXEC, 405
    parameter block, 534
    parameter block, 534
    parameter block, 533
executable types, 533
executables;
    .EXE file header, 534
    .EXP file header, 542
    .EXP file header, 542
    compressed, 534
    new format, 535
    Phar Lap, 542, 543
    self-extracting archives, 534
executing programs, 532, 543, 646, 912, 931-32, 1167
EXEPACK, 533
Expanded Memory Specification -- see EMS, 737
expansion bus;
    *see also EISA, Intel PCI BIOS, PCMCIA,*
    PCI, 67
    EISA, 65
expansion unit, 38
Explosiv, 291
    blanking interval, 292
    installation check, 292
    undetermined functions, 292-93
    uninstall, 292
extended memory;
    copy memory, 40
extended BIOS data area, 46
extended memory;
    size, 40
extended control register, 148
extended file attributes, 563
extended error information, 565
extended attribute properties list, 563
extended attribute value list, 563
extended BPB, 579
extended file attributes, 562
extended error information, 569
extended memory usage, 697
extended memory, 683, 696, 746, 767, 902

Extended File Control Block—see also FCB, 499
Extended Memory Specification -- see XMS, 737
Extended Batch Language, 1174
external monitor, 153
External Device Interface—see XDI, 879
EZRECV, 272

F-PROT;
    F-NET, 1075
    F-POPUP, 1077
    F_XCHK, 1076
    F-LOCK, 1076
F-DLOCK, 1077
F-PROT;
    F-DRIVER.SYS, 1075
F-LOCK, 1076
F-DRIVER.SYS, 1075
F-POPUP, 1077
F-PROT;
    F-DLOCK, 1077
F_XCHK, 1076
FAKEY;
    adding keystrokes, 282
    clearing keyboard buffer, 282
    installation check, 281
    sound, 282
    tone array, 282
Family API;
    JOIN, 622
    DosQFileInfo, 621
    DosQPathInfo, 621
    DosSetFileInfo, 622
    SUBST, 622
    DosSetPathInfo, 622
far pointer comparison, 591
Far East DOS, 574
    determining type, 596
FARTBELL, 483
    force noise, 483
    installation check, 483
    resident code segment, 483
FAST!, 337
    cache information, 338
    cache information, 338
    delayed writes, 337
    disabling cache, 337
    function numbers, 337
    hotkeys, 337
    installation check, 339
    mouse checks, 337
    request packet, 337-339
    resizing, 337
    uninstall, 339
FASTBUFF, 280
FastGraph/Light—see FGDRIVER, 987
FastJuice;
    installation check, 1170
    uninstall, 1170
FASTOPEN, 592
    entry point, 593
FAT, 512
faults;
    see exceptions, 21
FAX software;
    MTEZ XpressFax, 1122
    see also CAS, 1127
    ZFAX, 1124
    Gammafax DOS Dispatcher, 1125
    FAXPLUS, 1125
    BitFax Scheduler, 1109
    Comwave Microfax Specification, 1125
FaxBIOS, 1110
    commands, 1111
    function numbers, 1111
    installation check, 1111
FAXPLUS, 1125
FAXTSR, 1125
FCB tables, 553
FCB;
    structure, 499-508
FCBS=, 550
feature bytes, 43
feature bits, 151
February, 592
FFLUSH, 578
FGDRIVER, 987
FGDRIVER:video operations, 1010
FGDRIVER;
    attribute changing, 996
    available memory, 1010
    bitmap size, 1002
    bitmaps, 998
    circles, 995
    clearing screen, 988

  clipping region, 993
  color mapping, 999
  colors, 990, 991, 992, 1011
  copying screen regions, 1000
  copying video pages, 1001
  DAC registers, 990, 991, 1011
  delays, 1009
  EGA information, 988
  ellipses, 994
  error codes, 997
  fade-in/fade-out, 1002
  flood fill, 994
  graphics cursor, 993
  graphics cursor, 992
  graphics cursor, 992
  installation check, 987
  joystick, 1004
  keyboard input, 1003
  line drawing, 993
  logical video pages, 1010
  masking map, 1000
  memory management, 1011
  mode-independent bitmaps, 998
  mouse, 1007
  music, 1008
  palette, 992
  panning, 1002
  patterns, 999
  PCX files, 1002
  PCX files, 1001
  pixel run maps, 999
  pixel manipulation, 992
  pixel manipulation, 992
  polygons, 994
  reading screen, 996
  rectangles, 994, 1011
  rectangles, 1011
  screen output, 995
  screen size, 989
  screen position, 989
  scrolling, 1002
  shift keys, 1004
  sound, 1008
  text cursor, 995
  text attribute, 996
  text cursor, 988
  time, 1009
  transparent colors, 1001
  video pages, 996
  video page management, 1010
  video pages, 997
  video page management, 1010
  video mode, 987
  video page management, 997
Fidonet, 4
file output, 86
file attributes;
  Acorn BBC Master 512, 85
file input, 84
file compression;
  DIET, 321
  PCMANAGE/DCOMPRES, 334
  dLite, 334
file position, 398
file handles, 396
file types, 399
file attributes, 402
file date, 402
file sharing;
  retry count, 525
file sharing modes, 519
file pointer—see also file position, 531
file attributes;
  extended attributes, 627
  DR-DOS extended attributes, 614
file sharing hooks, 551
file attributes;
  extended attributes, 626
file handle table, 606
file position, 531
file owner, 615
file locking, 955
file backups, 1164 functions, 505
File Control Block functions, 499, 500, 501-2, 503, 505, 508
File Allocation Table, 512
filename parsing, 508
filename comparison, 592
filename terminator table, 576
files;
  attributes, 402
  canonical names, 572
  character device names, 399
  closing, 398
  commit, 579
  creating, 397
  drive for file, 591
  file pointer, 398
  file position, 508
  finding, 399
  inherited, 518
  normalizing filename, 591
  opening, 83
  random access, 505
  reading, 397
  renaming, 400
  resizing, 520
  size, 505
  status, 524
  temporary, 566
  timestamp, 562
  timestamp, 562
  truncating, 397
  writing, 398
  writing, 505
  writing, 520
  writing, 502
FILES=, 550
FILESAVE.INI file, 1102
fils;
  deleting, 400
findfirst, 399
FindFirst, 501
  data block, 545
finding file;
  via FCB, 501
  via FCB, 501
FindNext, 501
fixed disk—see hard disk, 28
flags;
  IF, 674
  IF, 674
  IF, 699
  IF, 674
Flambeaux Software, Inc., 1168
Flash-Up Windows, 174
FlashTek -- see also Phar Lap, 702
FlashTek X-32VM, 702
FlashTek;
  32-bit stack, 704
  available memory, 704
  calling real mode, 692
  calling real mode, 693
  error codes, 691
  exception handlers, 703
  exception handlers, 703
  IRQ mapping, 691
  memory allocation, 704
  memory mapping, 704
  paging, 702
  protected-mode addresses, 692
  real-mode interrupt vectors, 690
  real-mode callbacks, 692
  real-mode interrupt parameter block, 693
  real-mode addresses, 692
  reset DOS extender, 689
  resignalling interrupts, 690
  segment manipulation, 704
  selector allocation, 703
  selectors, 702
  virtual memory management, 702
FLASHUP.COM, 174
FlexOS, 613
FLIST, 279
floating-point emulation, 1021
FLOCK, 566
floppy disk;
  number installed, 31
  hardware interrupt, 28
  4+Power controller, 217
  3-mode drive, 216
  logical drives, 596
flow control, 256
FLU_SHOT+, 1076
FM.COM, 471
  background sounds, 473
  number of voices, 473
  undetermined functions, 473
FN32, 1144
FNINIT instruction, 23
fonts, 909
fonts, 909
  GRAFTABL, 635
  LCD, 131
  video, 131-141
FORCEDUP, 531
FOSSIL;
  video, 257
  cursor, 257
  break, 258
  character input, 256
  control-key checks, 257
  driver information, 259
  DTR control, 255
  external functions, 261
  flushing buffers, 255
  initialization, 254-259
  installation check, 254
  keyboard input, 256
  modem control register, 260
  MX5, 262-263
  output character, 256
  parity, 260
  port control, 260
  rebooting, 258
  serial input, 254-261
  speed, 260
  timer tick parameters, 255
  transmit control, 257
  watchdog, 257
  word length, 260
  X00, 259-260
Frank A. Bell, 285
Frans Veldman, 1078
Free Software Foundation, 733
Fridrik Skulason, 1077
Frieze;
  API, 296
  execute command, 299
  flushing buffers, 299
  image size, 298

  - image merging, 299
  - image file, 298
  - IN.SET file, 300
  - initialization, 298
  - installation check, 296
  - linked-mode routine, 299
  - linked mode, 299
  - menu, 299
  - paper movement, 299-300
  - parameter table, 298
  - scaled images, 300
  - setup, 300
- FROB.SYS, 3
- FSBBS, 1149
  - account name, 1149
  - configuration, 1149
  - dump mode, 1151
  - functions, 1150
  - PassData buffer, 1150
  - terminal number, 1150
  - timer, 1150
  - user record, 1149
- Future Domain;
  - formatting, 235
  - initialization, 238
  - disk information block, 239
  - disk information block, 239
  - device types, 237
  - stopping disk, 235
  - SCSI mode select, 234
  - peripheral qualifier, 237
  - medium capacity, 238
  - controller family code, 238
  - defect tables, 235
  - controller RAM, 239
  - controller information, 237
  - device information, 235-239
  - defect tables, 235
- Future Computing Systems, 337
- Futurus Team, 1177
- FWAIT, 1020
- G.T. Becker, 1161
- Gammafax DOS Dispatcher, 1125
- garbage collection, 1015
- GDI, 819
  - device classification, 820
  - information, 819
- GDT, 683
- GDT selectors, 902
- GDT—see global descriptor table, 40
- GDTR, 652
- GDTR register, 757
- GEM, 1030
- general protection violation, 24
- General Software, 627
- Genius Mouse;
  - copyright string, 452
  - storage requirements, 453
  - configuration, 453
  - enabling, 454
  - button assignments, 454
  - disabling, 455
  - acceleration level, 455
- Genius mouse driver;
  - number of buttons, 464
  - screen update, 464
  - screen update list, 464
  - see also mouse, 464
- Genoa 7900—see Tseng, 197
- Geoff Friesen, 478
- Geoff Chappell, 815
- George Adam Stanislav, 177
- George Adam Stanislav, 285
- GET EXECUTABLE PATH, 473
- Gisbert W. Selke, 281
- global descriptor table;
  - switching to protected mode,
  - for extended memory copy,
- GNU C++, 733
- GO32, 733
  - DOS extensions, 733
  - relocated IRQs, 734
  - video extensions, 733
  - video modes, 733
- GOLD, 1151
  - installation check, 1151
  - state, 1151
  - state, 1151
- gonkulator, 3
- GRAFIX— see Hercules GRAFIX, 157
- GRAFTABL, 634
  - font table, 635
  - graphics data, 621
  - installation check, 620-635
- GRAPHICS.COM, 635
  - installation check, 635
- graphics;
  - area fill, 433
  - bitmap buffer, 94
  - bitmaps, 998-1032
  - character output, 159, 382. 433
  - circle drawing, 995
  - clearing, 432
  - clipping, 380
  - clipping region, 993
  - cursor, 993
  - display line, 163
  - draw, 380-432
  - drawing function, 381
  - ellipse drawing, 994
  - fill mask, 379
  - initialization, 432
  - line type, 381
  - line drawing, 993
  - origin, 380
  - panning, 433
  - panning, 1002
  - polygon drawing, 994
  - printing, 1032
  - read pixel, 381, 433
  - reading pixels, 122
  - rectangle drawing, 993-1011
  - scrolling, 1002
  - setting pixels, 121
  - write pixel, 381-432
  - zooming, 433
- gray-scale, 126
- Greenview Data, 1167
- Greg Shenaut, 223
- GRIDLOC.EXE, 1144
- GSS Computer Graphics Interface, 179
- GSS*CGI, 179
  - APL*PLUS/PC, 1016
- Guenther Thiele, 483
- Gupta Technologies, 1022
- GWBASIC, 1030
- Halo88, 179
- Hangeul input mode, 574
- hard disk;
  - partition record, 59
  - get physical drive number, 52
  - hardware interrupt, 28-30
  - operating system indicator, 59
  - master boot record, 58
  - parking, 1172
- hardware interrupts;
  - IRQs, 27
  - IRQs, 27
  - hard disk, 28
  - IRQs, 25, 27, 28, 29, 30, 764, 825
- HDILOAD, 181
- Headl1024i, 148
- Headroom, 943
  - installation check, 943
  - resident code segment, 943
  - undetermined functions, 943
- HEART, 1055
- Heath-Zenith, 96
- Helix Software, 683, 783, 1143
- help systems;
  - HP 95LX, 429, 430
- Henson Scientific, Inc., 283
- Herb Rose, 930
- Hercules Graphics, 157-60, 168, 760
- Hewlett-Packard, 94, 804
- HGC, 157
- Hicolor DAC, 198
- HiColor video modes, 197
- HIDOS.SYS, 619
- high memory, 596, 644, 758, 804, 1106
- HIGHUMM.SYS;
  - entry point, 800
- HIL BIOS;
  - installation check, 95
- HILOAD, 617
- HIMEM.SYS, 746-47, 750, 797-98
- Ho-Ping Tseng, 284
- hooking interrupts, 973, 1105, 1041
- HOOKROM.SYS, 1105
  - hooked interrupts, 1105
- horizontal panning register, 123
- horizontal retrace, 196
- host-connected keyboard, 57
- hot spot, 449
- hotkeys;
  - AD-DOS, 288
  - After Dark, 288
  - AMIS, 974
  - Back & Forth, 929
  - Blank-It, 291
  - Buffit, 1145
  - Burnout Plus, 294
  - DESKTOP, 1098
  - FAST!, 337
  - JORJ, 1140
  - MDEBUG, 1048
  - MultiDOS Plus, 891
  - OPTIMA 1024 VGA, 199
  - PC Thuis Organizer Shell, 1159
  - PCM.COM, 476
  - Phantom2, 280
  - Phantom2, 279
  - PRINDIR, 316
  - Ross Wentworth POPUP library, 978
  - TaskMAX, 912
  - TesSeRact TSR interface, 984
  - ZIPKEY, 1084
- Houston Instruments HiPad, 1147
- HP 95LX;
  - system shutdown, 36
- HP Vectra, 94, 95, 96
- HP 95LX;
  - suspend/resume, 95
- HP Vectra Extended BIOS, 148
- HP Extended BIOS, 147, 148
- HP MultiMode Video Display Adapter, 148
- HP 95LX System Manager;
  - Lotus 1-2-3, 384
- HP 95LX;
  - video mode, 379
  - system services, 379

    System Manager, 383-85, 390-91
    clip region, 380
    draw, 380
    pen color, 381
    line type, 381
    image buffer, 382
    pen movement, 381
    read pixel, 381
    save/restore image, 382
    graphics origin, 380
    graphics information, 379
    draw, 381
    drawing function, 381
    graphical text, 382
    graphics information, 379
    graphics fill mask, 379
HP 95LX System Manager;
    screen services, 391
    event wait, 383
    event record, 383
    event wait, 384
    file list workspace, 388
    file selection menus, 387
    screen services, 390
    error codes, 388
    editing services, 393
    clearing screen, 389
    bridge services, 384
    cursor position, 391
    display string, 389
    editing services, 392
    edit record, 392
    file selection menus, 388
    file menu structure, 387
    file selection menus, 388
    multi-line edit record, 392
    menu system, 387
    screen services, 389-390
    menu system, 387
    screen services, 389
    flush keyboard buffer, 385
    keyboard, 387
    file selection menus, 389
    menu system, 386
    menu data, 386
    communications services, 419-420
    communications reset options,
    communications services, 420
    clock/calendar services, 416-417
    collation services, 430
    clock/calendar services, 415-423
HP 100LX;
    memory mapping, 435
HP 95LX System Manager;
    BIOS output, 427
    beep, 408
    application name, 406
    character output, 433
    clipboard services, 406
    alarm record, 411
    "CP_TO_LICS", 434
    communications services, 424
HP 100LX;
    memory mapping, 435
HP 95LX System Manager;
    "M_FORM_FT", 428
    "M_RAM_IV_INFO", 428
    alarm, 417
    "SHOWNAME", 425
    clipboard services, 407-408
    clock/calendar services, 411-425
    clipboard services, 408
    communications services, 425
    printer services, 417-418
    popup boxes, 426
    process management, 404-405
    popup boxes, 425
    pixels, 432
    help, 429-430
    graphics services, 433-434
    help, 430
    keyboard, 393
    memory services, 409-410
    macros, 427
    keyboard types, 414
    macros, 427
    process management, 405-406
    process management, 406
    timer address, 426
    timers, 415
    time-related settings, 414
    time parsing, 417
    time parsing rules, 417
HP 95LX;
    calculator, 435
    error codes, 406
    alarm, 411
    alarm, 410
HP 95LX System Manager;
    timestamp, 412
    time parsing, 416
    time format, 414
    resource services, 429
    screen updates, 428
    resource services, 429
    process management, 406
    rename file, 400
    separator characters, 428
    sound services, 408

    system settings, 415
    time display, 415
    system resources, 426
    sound services, 409
    graphics services, 433
    editing services, 394
    error message, 425
    file attributes, 402
    file services, 397
    file I/O, 398
    file date, 402
    editing services, 394
    current directory, 400
    data segment, 434
    date format, 413
    current directory, 400
    date parsing, 416
    directory management, 401
    editing services, 393
    directory management, 401
    default drive, 402
    date parsing rules, 416
    default drive, 401
    file services, 397-403
    file state record, 396
    graphics save/restore, 434
    graphics services, 433
    graphics save/restore, 434
    file services, 399-402
HP;
    100LX palmtop—*see also HP 95LX*, 94
    95LX palmtop—*see also HP 95LX*, 94
    HIL—*see also HIL BIOS*, 95
HPMM.SYS, 365
HRAM, 804
HRAMDEV.SYS, 804
    installation check, 804
Human Interface Link;
    *see Hewlett-Packard, 94*
Hummingboard DOS extender, 707
HyperDisk, 215
    cache state, 341
    caching flags, 340
    installation check, 340
    resident code segment, 340
    undetermined functions, 340
HyperKey;
    installation check, 340
HyperPAD, 1176
HyperScreen;
    installation check, 340
HyperStb;
    installation check, 340
hypertext, 1168
HyperWare, 339
    product codes, 340
I/O ports;
    trapping, 687-88, 791
I/O ports;
    QEMM-386, 763
I.V. Morozov, 165
IBM Convertible;
    system status flags, 37
    power management, 37
    system status, 37
    system shutdown, 36
    profiles, 36
    external event wait, 36
    power management, 36
    suspend/resume, 63
IBM SCSI interface, 245
IBM Graphic Development Toolkit, 179
IBM/Yale EBIOS—see EBIOS, 268
IBM Digitized Sound Package—see
        IBMSND driver,
IBM Genie, 1143
IBMAVSH, 1076
IBMCACHE.SYS, 360
    error list, 361
    status record, 360
IBMDOS.COM, 590
IBMSND driver, 480, 481, 482
IC, 478
ICCTSR, 182
IDE controllers;
    ADP-60, 218
idle, 373
idle functions, 476
idle interrupt, 600
IDT, 41, 700, 709
IDTR register, 757
IF register;
    virtualized, 674, 699
IFS;
    IOCTL, 579
Image Processing Interface, 1031, 1032,
        1033, 1034
ImageCapture, 182

IN.SET file, 300
index pcANYWHERE;
    termination request, 1064
InDOS flag, 513
initialization files;
    mouse, 452
InnerMission, 294
Innovative Data Concepts, 983
InPort mouse, 446
input from file, 83, 84
input;

  from serial port, 247, 251, 254, 258, 261, 265, 267, 274, 898, 901, 948-49
  from HP 95LX communications channel, 422
  from standard input, 496, 497, 498
  from keyboard, 496, 497, 903, 1003, 1159
  from AUX device, 925
  from screen, 996
INSET, 298
  installation check, 298
INSTALL=, 550
installable commands;
  execute, 644
  installation check, 644
installation check;
  EISA System ROM, 65
  PCI, 67
  HP 95LX, 94
  Genoa video cards, 100
  HP HIL BIOS, 95
  ATI video cards, 99
  Ahead video cards, 99
  FLASHUP.COM, 174
  FOSSIL, 254
  M10_SCR, 164
  Common Access Method SCSI interface,
  Digiboard DigiCHANNEL, 250
  EGA Register Interface Library,
  Common Access Method, 230
  xDISK, 219
  TARGA.DEV, 230
  SPEEDSCR.COM, 174
  SCSILink, 240
  MSHERC, 168
  UltraVision, 160
  SCROLOCK, 165
  VESA, 183
  SCREENR, 165
  EGAPAL.COM , 174
  CD-ROM redirector, 244
  VGAPAL.COM , 174
  VUIMAGE display driver, 163
  AVATAR.SYS, 177
  BNU FOSSIL, 254
  ATI SMS.COM, 166
  ANSIPLUS.SYS, 177
  ADAPT, 167
  NewSpace, 330
  MX5 Extended FOSSIL, 263
  Newkey, 285
  MBBIOS, 264
  PCSpool, 312
  PCED, 278
  PC Paint Plus, 318
  Phantom of the Keyboard, 279
  LaserPort, 319
  KEYDEF, 278
  LPTx, 316
  LPTx, 316
  LPTx, 316
  M16_KBD.COM, 281
  KEYBUI, 280
  TIGA Communications Driver, 273
  TFPCX, 274
  TSRCOMM, 269
  VGAsave, 294
  WCED, 283
  VSTACK, 278
  TEXTCAP, 281
  TEMPLEXX, 283
  PRINT, 314
  PRINDIR, 317
  SCRNSAV2, 294
  STACKEY, 282
  StuffIt, 284
  PIPELINE, 273
  LPTx, 316
  Blank-It, 290
  ATTRIB, 278
  CED, 283
  COURIERS.COM, 266
  DIET, 321-324
  ASSOC, 279
  AQUEDUCT, 273
  4MAP, 284
  2XON, 332
  AD-DOS, 287
  After Dark, 287
  ANARKEY, 275
  Disk Spool II, 306
  FLIST, 279
  FASTBUFF, 280
  Frieze, 296
  IBM/Yale EBIOS, 268
  INSET, 298
  InnerMission, 294
  Jetstream, 300
  FAKEY, 281
  Disk Spool II, 306
  Explosiv, 292
  DOSED, 283
  DOSKEY, 277
  DoubleSpace, 325
  Smart Prompt, 363
  SmartDrive, 352
  Super PC-Kwik, 359
  Virtual DMA Specification, 365
  Qualitas Qcache, 359
  DUBLDISK.SYS, 333
  FAST!, 339
  HyperDisk, 340
  CUBIT, 334
  COMBI-disk, 361
  APM, 371
  CACHECLK, 361
  QuickCache II, 345
  HyperKey, 340
  PCMANAGE/DCOMPRES, 334
  POWER.EXE, 376
  PC-CACHE.SYS, 359
  PC-Cache, 359
  HyperStb, 340
  PC-Cache, 343
  HyperScreen, 340
  Logitech mouse driver, 440
  Smooth Mouse Driver, 461
  SND, 478
  RESPLAY, 480
  Quick Mouse Reset, 462
  ZWmous, 463
  Z-Nix mouse driver, 464
  SoundBlaster speech driver, 467
  digital sound, 477
  IBMSND driver, 480
  Ad Lib SOUND.COM, 465
  Media Vision PCM.COM, 474
  MVSOUND.SYS, 472
  POPUP.COM, 456
  PCM driver, 469
  PrecisePoint, 461
  MAGic, 488
  Tinytalk Personal, 490
  MIDI driver, 482
  FARTBELL, 483
  AutoBraille, 486
  ASAP, 485
  MS-DOS internal utility functions,
  DRIVER.SYS support, 582
  ASSIGN, 632
  4DOS, 645
  critical error handler, 632
  European MS-DOS 4.0, 606
  European MS-DOS POPUP, 610
  DR-DOS, 616
  Concurrent DOS, 615
  APPEND, 630
  ANSI.SYS, 629
  DISPLAY.SYS, 633
  NLSFUNC, 637
  NDOS, 645
  SHARE, 640
  VCPI, 649
  KEYB, 621
  XMA2EMS, 641
  KSTACK, 646
  KEYB, 635
  DOS Protected-Mode Interface,
  DISPLAY.SYS, 634
  GRAFTABL, 634
  GRAPHICS, 635
  installable commands, 644
  GRAPHICS.COM, 635
  GRAFTABL, 620
  Ergo DOS extenders, 707
  OS/x86, 705
  Hummingboard DOS extender, 707
  DOS/16M, 709
  Phar Lap 386|DOS-extender, 701
  DOS/4GW, 710
  DPMS, 679
  Phar Lap DOS extenders, 701
  EGA.SYS, 817
  EMM386.EXE, 799
  MICEMM, 786
  HRAMDEV.SYS, 804
  MS Windows, 805
  QEMM-386, 753
  Netroom PRENET, 783
  QEMM-386, 754
  RTM, 725
  XMS, 747
  TKERNEL, 726
  Netroom POSTNET, 783
  Netroom, 783
  DOS/4G, 713
  AI Architects, 735
  DPMILOAD, 715
  DPMILOAD, 721
  MEMLIMIT, 782
  MS Windows, 808
  XLOAD, 792
  XMS version 1, 805
  WINX, 821
  WINOLDAP, 817
  MS Windows, 812
  RM386, 786
  DVSIXDI, 884
  DVTXDI, 884
  TaskMAX, 911
  DVDOS4GX.DVR, 874
  QEXT.SYS, 883
  DESQview, 873
  Back & Forth, 929
  DESQ, 883
  VMiX, 901
  DESQview XDI, 879
  PC-Mix, 969
  NOTE, 972
  PC-MOS/386, 950

  NOLPT, 972
  MultiLink Advanced, 970
  Switch-It, 969
  SWELL.EXE, 965
  SPELLER, 972
  Software Carousel, 962
  CTask, 969
  CPTASK, 961
  AMIS, 972
  Alternate Multiplex Interrupt Specification,
  Task Switcher, 937
  DOSSHELL, 933
  Headroom, 943
  FASTMOUS, 972
  DoubleDOS, 940
  Monitor, 972
  RECALL, 972
  Ross Wentworth POPUP library, 978
  SoftCom programs, 985
  RBkeyswp, 972
  Personal Calendar, 985
  Microsoft Profiler, 1035
  Microsoft SQL Server, 1025
  SWITCHAR, 972
installation checks;
  TurboPower TSRs, 978
  VGABLANK, 972
  XPTR2, 972
  TurboPower TSRs, 977
  TSRBONES, 977
  TesSeRact TSR interface, 982
  ThesPlus, 985
  Kingswood TSRs, 980
  MicroHelp Stay-Res, 985
  Btrieve Multi-User, 1019
  CAL, 985
  CAPDOS, 1039
  Btrieve, 1026
  BorlC++ IDE, 1019
  BMB Compuscience Canada Utilities,
  BorlC++ IDE, 1019
  Kingswood TSRs, 980
  CappaCom programs, 985
  CiriSOFT TSR interface, 979
  FGDRIVER, 987
  Image Processing Interface, 1031
  FastGraph/Light, 987
  Compaq UILIB, 1025
  EATMEM, 972
  DESKTOP, 1097
  DOS-UP.SYS, 1105
  DATAMON, 1089
  EBL, 1174
  DATEFIX, 1140
  ELRES, 1141
  F-POPUP, 1077
  F-XCHK, 1076
  F-LOCK, 1076
  F-DLOCK, 1077
  ELRES, 1142
  Extended Batch Language, 1174
  FastJuice, 1170
  CUCKOO, 1145
  ThunderByte, 1078
  THELP, 1144
  TeleReplica, 1065
  TMED, 1139
  TRAP, 1055
  Turbo Lightning, 1171
  Turbo Debugger, 1054
  TBSCANX, 1078
  TAME, 1141
  SCROLLit, 1143
  SCOUT, 1170
  SFENGINE, 1122
  SitBack, 1164
  StopPrg, 1055
  StackMan, 1165
  VIDCLOCK.COM, 1143
  VIDRAM, 1107
  WATCH.COM, 1139-1143
  WHOA!, 1167
  WILDUNIX.COM, 1143
  XpressFax, 1122
  Word Perfect Third Party Interface,
  VWATCH, 1074
  VSWAP, 1167
  viruses, 1072
  VSAFE, 1074
  ScanBoot, 1074
  REDVIEW, 1144
  JORJ, 1140
  IBMAVSH, 1076
  Jot-It!, 1151
  MAKEY, 1140
  METZTSR, 1154
  MANIFEST, 1106
  IBM Genie, 1143
  HyperPAD, 1176
  FN32, 1144
  FLU_SHOT+, 1076
  Futurus Team, 1143
  GOLD, 1151
  HEART, 1055
  GRIDLOC, 1144
  Microsoft Word TSR cooperation,
  MIN-MEM, 1144
  PUSHDIR.COM, 1140
  PCW Weather Card, 1161
  QDPMI, 1108
  Quarterdeck RPCI, 1105
  RainboxFAX, 1122
  RAID, 1161
  PCShell, 1095
  PCRUN, 1096
  Nabbit, 1144
  MTEZ XpressFax, 1122
  NJFRERAM, 1145
  Norton Guides, 1101
  PCL-838, 1154
  PARKER, 1139
  FaxBIOS, 1111
  XVIEW, 1168
  pcANYWHERE IV/LAN, 1062
  ASCII, 1146
  SETDRVER, 1163
  Norton Utilities, 1101
  CPSCHED, 1086
  ZyXEL ZFAX, 1124
  ZIPKEY, 1081
  AUTOPARK.COM, 1172
  Communicating Applications Specification,
  BACKTALK, 1085
  CPSCHED, 1086
  CALCULATOR, 1140
  COMMUTE, 1060
  Buffit, 1145
  BitFax Scheduler, 1109
  BorlTDX, 1054
  Paradise video, 99
  Tseng ET4000, 99
  Oak Technologies, 99
instance data;
  MS Windows, 806
  Task Switcher, 939
instancing, 815
  installation check, 1041
  INTCFG;
  see also INTMON, 1041
  INTCFG;
  INTCFG;
  uninstall, 1042
  Intel PCI BIOS;
  protected-mode entry point, 67
Intel PCI BIOS;
  installation check, 67
  hardware characteristics, 67
  bus operations, 68-71
  bus operations, 71
  error codes, 68
  attached devices, 71
  32-bit calls, 71
Intel, 67
  32-bit calls, 71
  attached devices, 68
Intel Code Builder;
  installation check, 701
Intel Image Processing Interface—see Image Processing
Intel SatisFAXtion, 1138
interrupt vectors, 2
interrupt request, 2
interrupt descriptor table, 41
interrupt controller, 29
interrupt controller;
  Z100, 97
interrupt controller;
  Z100, 97
interrupt vectors;
  getting, 514
  setting, 506
interrupt vectors;
  protected mode, 661
  DPMI, 660
interrupt controller;
  vectors, 651
  DPMI, 660
  protected mode, 662
  vectors, 687
  vectors, 691
  real mode, 690
  Netroom, 783
  protected-mode, 724
  QEMM-386, 759
  Netroom, 783
  RM386, 786
interrupt sharing protocol, 971
interrupt hooking, 973
Interrupt List, 4
interrupts;
  reserved, 2
INTMON, 1041
INTMON;
  CPU registers, 1041
INTMON;
  hook interrupt, 1041
INTMON;
  installation check, 1041
INTMON;
  reset, 1041
INTR, 2
INTRSPY, 1056
INTRSPY;
  entry point, 1056
invoking interrupts, 908
IO.SYS, 806
IOCTL capabilities, 530
IOCTL call;

  generic, 620
IOCTL calls;
  ANSI.SYS, 629
  Compaq AG1024.SYS, 176
  generic requests, 525
  IFS drives, 579
Iomega, 216
IRQ mapping, 41
IRQ usage;
  mouse, 447
IRQ channels;
  MVSOUND.SYS, 472
IRQ mapping, 651, 687, 691
IRQ relocation, 734, 873, 942, 1162
IRQ reservation, 956
IRQs, 2, 25, 27-30, 266
IRQs;
  relocated, 627
Isogon Corporation, 329
ISR, 1057
J.M. Allen Creations, 317
Jacob Rieper, 1163
James W. Birdsall, 273
James Berry, 1064
Japanese, 77
Japanese;
  AX PC, 77
Japanese;
  character conversion, 79
Japanese;
  printer, 79, 80
Jetstream, 300-301
JFT, 592, 623, 606
Jim Harper, 244, 245
JIS, 79
Job File Table, 578, 592, 623
Johan Zwiekhorst, 280, 1139
John J. Seal, 164
John W. Ratcliff, 480
John Vanderaart, 1155
John Socha, 1101
JOIN, 555
JOIN;
  OS/2, 622
Jorj Software Co., 1140
JORJ, 1140
Jot-It!;
  uninstall, 1152
  user name, 1151
  installation check, 1151
  message directory, 1152
joystick, 39, 1004
JP Software, Inc., 645
JPI;
  TopSPEED Modula-2, 1025
Kana Kanji Converter, 1172
KANSYS, Inc., 486
KBUF;
  adding keys, 282
Kermit, 1151
Kevin Stokes, 294
key stuffer, 284
Key Software Products, 800
KEYB, 55
KEYB, 635
KEYB;
  called unknown functions, 641
  called unknown functions, 641
  code page table list, 636
  configuration, 621
  installation check, 635
  installation check, 621
  keyboard code page, 636
  keyboard mapping, 636, 637
  resident code segment, 621
KEYB;
  translation data, 636
keyboard intercept, 37
keyboard type, 93
keyboard ID, 56
keyboard layout, 621
Keyboard FOSSIL, 281
keyboard;
  122-key, 57
keyboard;
  122-key, 57
  adding to buffer, 55, 282-284, 866-869, 940, 1017
  FOSSIL, 256
  HP 95LX menus, 387
  HP 95LX, 393
  KSTACK, 646
  CapsLock, 1003
  clearing buffer, 282
  clearing typeahead, 940
  Compaq, 93
  control-break, 62
  Cyrillic, 281
  data ready interrupt, 25
  DESQview special keys, 55
  enhanced, 56
  flush buffer, 385
  get keystroke, 53, 56
  get functionality, 55
  get keystroke, 57, 80, 285
  host-connected, 57
  HP 95LX, 94
  initialization, 80
  input from, 256
  input line, 277
  key translation, 285
  key bindings, 284
  key translation, 284
  keyclick, 54
  keystroke availability, 53, 56, 57
  macros, 285
  mapping, 636, 637
  mode, 957
  NEC PC-9800, 80
  new input, 287
  NumLock, 1004
  PCjr, 55, 284, 285
  print screen, 31
  scan codes, 26
  scan code translation, 37
  ScrollLock, 304, 1004
  select layout, 55
  shift flags, 57
  shift keys, 79
  special keys, 26
  special keystrokes, 55
  stuffing buffer, 646
  supported functions, 55
  templates, 283
  typematic rate, 54
  Z100, 97
KEYBUI, 280
keyclick, 54
KEYDEF, 278
keypresses, 80
KEYSTACK.SYS, 523
Kim Kokkonen, 1139
Kingswood Software, 980-81, 1152-53
Kofax KF9X00, 1174
Kraft Mouse;
  see also mouse, 464
Kraft Mouse;
  undetermined function, 464
Kristofer Sweger, 177
KSP$UMM device, 800
KSTACK;
  installation check, 646
KSTACK;
  stuff keys, 646
KYE Corp., 452
Kyle, Jim, 4
LA$TBYTE device, 800
LANtastic shared resources, 545
LapLink Quick Connect, 272
LaserPort;
  execute function, 320
LaserPort;
  installation check, 319
Lassen Software, Inc., 1166
LASTBYTE.SYS, 800
LASTBYTE.SYS;
  see also HIGHUMM.SYS, 800
LASTDRIVE, 498, 550
LASTDRIVE;
  OS/2, 623
LCD attributes, 132
LCD fonts, 131, 132
LCD display, 154
LDT, 695, 721, 812
LDT;
  Phar Lap, 694
LDTR, 652
Liang Chen, 1139
light pen, 97, 118, 147, 443
line buffer, 277
line input, 497
linear executable;
  fixups, 541
  format, 537
  CPU types, 537
  entry points, 540
  module types, 538
linear executables, 533
  page map table, 539
  object flags, 539
List of Lists, 547
LOAD32.EXP, 875
loading programs, 694
loading programs high, 792
local bus;
  VESA, 67
logical drives, 530
logical drives, 530
logical drives;
  change notification, 596
Logitech mouse driver, 440, 457-58, 459-61
Logitech Modula, 1029
Logitech;
  button presses, 441
Lotus 1-2-3, 384, 709
LPT interrupts, 27
LPT1, 28
LPTx;
  installation check, 316
LZS compression, 331
M10_SCR;
  installation check, 164
M16_KBD, 281
Mach 20, 798
macros;
  HP 95LX, 427
MAGic, 488-90
mailboxes, 830, 849-50, 888
  finding, 865
  locking, 856
  name, 861

  names, 854
  special names, 865
MAKEY, 1140
MANIFEST, 1106
mapping context;
  MultiDOS Plus, 893
mapping context;
  MultiDOS Plus, 894
mapping context;
  MultiDOS Plus, 893
Marek Kosznik, 1140
Mark Adler, 282
Mark J. Cox, 479
master boot record, 58
MBBIOS;
  break, 265
  character input, 265
  DTR control, 264
  DTR control, 264
  initialization, 264
  installation check, 264
  options, 265
  serial input, 265
  serial output, 265
Memory Control Block (MCB), 549
  DOS data subsegment, 550
  DR-DOS, 618
MD386, 802
  alternate map structure, 802
  alternate register maps, 803
  breakpoints, 802
  debugging registers, 803
  handle table, 802
  initialization, 802
  memory protection, 803
MDEBUG;
  callouts, 1042-1044
  commdriver, 1046-1047
  configuration, 1047
  display driver, 1043
  driver identification table, 1047
  error codes, 1049
  error codes, 1045
  flags, 1049
  graphic data, 1043
  graphic data, 1043
  help registers, 1048
  identification table, 1048
  initialization, 1044
  monitor, 1046
  original INT 2Fh, 1047
  password, 1047
  popping up, 1049
  resident code segment, 1047
  start/end driver, 1047
  status, 1042
  text screen, 1043
Media Vision, 469
Media Vision PCM.COM;
  "shark" functions., 474
Media Vision;
  see also FM.COM, PCM.COM, MVSOUND.SYS,
Media Vision;
  number of voices, 473
Media Vision;
MEMLIMIT;
  API, 782
  installation check, 782
  request packet, 782
  return codes, 782
MEMMAX, 619
memory size, 32
memory managers;
  Exception #13, 24
memory map;
  table, 48
memory managers;
  Exception #12, 24
memory access;
  aligned, 24
memory allocation, 409-435
  DPMI, 659
  European MS-DOS, 604
  HP 95LX, 409-532
  MS-DOS, 531-532
  Phar Lap, 691-700
  shared memory, 676
  strategy, 564
  VCPI, 650
memory available;
  DPMI, 671
  FlashTek X-32VM, 704
memory cache;
  external, 48
  PS/2 error codes, 48
memory chain;
  UMBs, 565
memory map;
  DPMI, 673
  EMS, 739-746
  FlashTek, 704
  get information, 47
  HP 100LX, 435
  HP 100LX, 435
  HP 95LX, 409
  physical address, 650
  QEMM-386, 756
  VCPI, 650
  Phar Lap, 699
memory size;
  Cirrus Logic, 194
  video, 197
memory;
  copying, 729
  extended, 728
  speed, 775
menus;
  HP 95LX, 387
metafile, 818
METZTSR, 1153
METZTSR;
  Ctrl-Alt-Del, 1154
  installation check, 1154
  keyboard activity, 1154
MICEMM, 755
  address map, 786
  installation check, 786
  linear address, 785
  memory types, 786
Micro Channel, 365
Microchannel bus;
  programmable option select, 47
Microchannel bus;
  programmable option select, 47
Microchannel bus;
  presence check, 43
MicroECLIPSE, 1174
MicroHelp Stay-Res;
  uninstall, 985
MicroHelp Stay-Res;
  installation check, 985
MicroHelp Stay-Res Plus, 986
MicroHelp Stay-Res, 985
Micronics, 754
Micronics Expanded Memory Manager—see MICEMM, 783
Micronics, 785
Microsoft Real-Time Compression Interface—see
  undetermined functions, 453
  see also mouse, 439
  undetermined functions, 453
  version number, 453
Microsoft Mouse;
  resetting, 440
  button states, 440
  setup, 1035
Microsoft Profiler;
  SEGDEBUG interface, 1036
Microsoft Profiler;
  setup, 1035
Microsoft SQL Server, 1025
Microsoft TSR Specification, 977
Microsoft Profiler;
  profiling, 1036
Microsoft Profiler, 1035
  profiling, 1036
  installation check, 1035
  profiling, 1036
  profiling, 1036
  profiling, 1036
Microsoft Word, 1139
Microsoft BASIC Compiler, 1172
Microsoft;
  EMM386—see EMM386, 797
  HIMEM.SYS—see HIMEM.SYS, 797
  MS Windows, 805
Microsystems Software, Inc., 488
MicroTalk, 485
MIDI;
  IBM Digitized Sound Package, 482
  installation check, 482
  Roland, 27
Mike Weaver, 884
Mike Blaszczak, 1145
MIN-MEM, 1144
Minix, 969
Mitch Davis, 1055
MKDIR, 401
modem control register, 249
modem;
  Compaq SLT/286, 91
  Compaq SLT/286, 90
Modula-2, 1029
monitor type, 132, 133-196
monitor;
  parameters, 132
monitors;
  switching between, 130
mounting drives, 326
mounting drives, 326
mouse, 437
mouse pointer, 441
mouse pointer, 1006
mouse cursor, 1099
Mouse Systems, 441-460, 813, 1005-1007
  driver state, 455
Mouse Systems;
  enabling, 454
  number of buttons, 456
  acceleration level, 455
  driver state, 455
  button assignments, 457
  disabling, 455
  POPUP.COM, 456
  Ultrares, 455
  undetermined functions, 454-457
  Ultrares, 455
  undetermined functions, 454
  button assignments, 454
  undetermined functions, 456

  see also mouse, 453
  storage requirements, 453
  configuration data, 453
  configuration, 453
  button presses, 441
MOUSEPWR.COM, 463
MRCI;
  capabilities, 328
  entry point, 329
  error codes, 329
  MRCINFO structure, 328
  MRCREQUEST structure, 329
  RAM-based server, 329
  ROM-based server, 328
MS-DOS;
  reentrancy problems, 257
  DoubleSpace, 324
  re-entering, 312
  exit code, 359
  directory manipulation, 517
  disk buffers, 498
  disk reset, 498
  Disk Transfer Area (DTA), 503
  directory manipulation, 517
  error codes, 505
  extended break checking, 512
  error codes, 493
  environment, 507
  Drive Parameter Block, 512
  environment, 506
  Drive Parameter Block, 504
  FCB functions, 499
  country code, 517
  country-specific information, 515
  commandline, 507
  closing files, 519
  character I/O, 498
MS-DOS;
  closing files, 500
  CP/M compatibility, 503
  default drive, 498
  CP/M compatibility, 504
  directory entry, 501
  FCB functions, 503
  reading files, 508
  PSP, 506
  kernel, 493
  Job File Table, 506
  OEM numbers, 511
  opening files, 499
  opening files, 518
  SETVER, 507
  switch character, 514
  version, 513
  writing files, 502
  terminate stay resident, 511
  Job File Table, 506
  File Control Block, 499
  finding files, 501
  filename parsing, 508
  finding files, 501
  InDOS flag, 513
  filename parsing, 508
  file size, 505
  file position, 505
  File Control Block, 499
  file position, 505
  FCB functions, 500
  character I/O, 496
  bugs, 506
  boot drive, 513
  AVAILDEV, 515
  character I/O, 496
  capitalization, 576
  canonical filenames, 572
  bugs, 578
  bugs, 577
  code pages, 577
  bugs, 566
  code pages, 577
  closing files, 590
  capitalization, 577
  ROM scan, 581
  exclusion list, 581
  ROM programs, 580
  bugs, 533
  BIOS Parameter Block, 560
  bugs, 545
  code pages, 577
  device status, 524
  device information, 523
  devices, 524
  file attributes, 521
  file attributes, 521
  device information, 522
  bugs, 520
  deleting files, 520
  file handles, 531
  IOCTL capabilities, 530
  logical drives, 530
  remote devices, 524
  IOCTL, 523
  file position, 521
  generic IOCTL, 525
  remote files, 524
  commit file, 568
  normalizing filenames, 591
  network support, 571
  memory control block, 549
  memory allocation strategy, 564
  OEM functions, 581
  OEM INT 21h handler, 581
  process ID, 547
  path separator, 590
  opening files, 580
  memory allocation, 532
  memory allocation, 532
  finding files, 546
  file timestamp, 562
  memory allocation, 531
  List of Lists, 547
  Job File Table, 578
  process ID, 573
  program termination, 544
  Swappable Data Area, 570
  special program names, 560
  temporary file, 566
  yes/no response, 576
  verify flag, 561
  UMB linkage, 565
  timestamp, 562
  SFT reference count, 590
  Set Handle Count call, 578
  PSP segment, 573
  PSP, 561
  record locking, 566
  renaming files, 561
  Set Execution State call, 543
  server function call, 567
  serial numbers, 578
  return code for program, 544
  extended file attributes, 564
  extended file attributes, 563
  disk buffers, 559
  double-byte characters, 573
  DOS Parameter List, 567
  disk buffers, 590
  creating files, 566
  country-specific information, 575
  commit file, 579
  device driver lookahead, 574
  datestamp, 562
  data segment, 590
  current directory, 531
  executing programs, 543
  executing programs, 532
  ERRORLEVEL, 544
  error locus, 566
  execution state, 544
  extended BPB information, 579
  extended file attributes, 562
  error class, 565
  enable/disable drive, 572
  Drive Parameter Block, 560
  drive data table, 588
  double-byte characters, 574
  DRIVER.SYS support, 582
  enable/disable drive, 571
  XMA2EMS, 640
  SHARE, 638
  NLSFUNC, 637
  critical error handler, 632
  DISPLAY.SYS, 633
  ASSIGN, 632
  GRAFTABL, 634
  GRAPHICS, 635
  KEYB, 635
  extended error information, 603
  floppy-disk change, 596
  HMA allocation, 596
  filename comparison, 592
  file limit, 606
  Far East DOS, 596
  fast console output, 601
  Job File Table, 592
  logical drives, 596
  version number, 595
  terminate stay resident, 600
  see also European MS-DOS, 603
  program termination, 597
  return codes, 607
  extended error information, 594
  idle interrupt, 600
  compatibles, 613
  checksumming memory, 592
  CP/M compatibility, 601
  device chain, 593
  critical error handler, 597
  bugs, 601
  APPEND, 630
  error tables, 594
  absolute disk read, 599
  bugs, 599
  disk interrupt handler, 595
  ANSI.SYS, 629
  error tables, 594
MS Windows;
  QEMM-386, 763
  TSR communication structure, 808
  switching virtual machines, 811
  TSR ID block, 809
  Startup Information Structure, 807
  selectors, 812
  video support, 816
  video, 816
  version check, 808
  video virtualization, 817
  installation check, 805
  initialization broadcast, 808
  installation check, 812
  instance data, 807
  restarting, 813

    ROM detection, 809
    initialization broadcast, 806
    WINOLDAP, 817
    WINOLDAP, 817-819
    VxD identifier, 810
    virtual devices, 807
    Virtual Device Reference Data, 807
    virtual device drivers, 813
    virtual machines, 809
    VxD, 813
    virtual device capabilities, 812
    critical sections, 816
MS Windows;
    callouts, 808-816
    instance data, 806
    switching virtual machines, 812
    instance data, 806
    background DOS, 816
    callouts, 806
    background DOS, 816
    clipboard, 819
    enhanced mode, 805
    EGA.SYS, 817
    EGA.SYS, 817
    entry point, 805
    entry points, 810
    ExitWindows, 813
    exit broadcast, 807
    DOSMGR, 814
    DOS application switch, 813
    CPU idle, 812
    clipboard, 819
    critical sections, 809
    critical sections, 809
    WINX, 821
    critical sections, 816
    identifying TSRs, 808
    debugging kernel, 1056
    debugging kernel, 1056
MSDOS.SYS, 590
MSHERC;
    installation check, 168
MSKK, 1172
MTEZ XpressFax, 1122
MTx device, 223
MultiDOS Plus, 802
    I/O drivers, 896
    hotkey, 891
    initialization, 893
    IRQ mapping, 893
    event triggers, 895
    executing commands, 890
    hotkey, 891
    threads, 889
    task termination, 889
    task control block, 891
    event counters, 895
    threads, 889
    time slice interval, 889
    IRQ mapping, 893
    error codes, 888
    critical sections, 890
    CPU idle, 887
    context-switching, 895
    background processing, 892
    EMS map, 893
    error codes, 897
    event counters, 895
    task control block, 891
    error codes, 887
    EMS map, 894
    event counters, 895
    IRQ mapping, 894
    serial buffers, 898
    serial monitor mode, 898
    serial port initialization, 897
    serial port status, 896
    serial port initialization, 896
    serial output, 898
    messages, 888
    memory start address, 895
    mailboxes, 888
    system block, 893
    mailboxes, 888
    messages, 888
    multitasking, 895
    semaphores, 887
    screen output, 898
    priority, 889
    serial port status, 897
    mailboxes, 888
    shadow screen, 890
    suspending tasks, 888
    system block, 893
    serial port status, 897
MultiLink Advanced, 970
multiple processors;
    Z100, 97
multiple processors;
    Compaq, 92
multitaskers, 929
    device usage, 38
    device usage, 39
    I/O waits, 42
    Cswitch, 930
    CTask, 969
    DESQview, 823
    DESQview/X, 823
    DoubleDOS, 939
    DR Multiuser DOS, 917
    DRMDOS, 917
    Minix, 969
    MultiDOS Plus, 887
    MultiLink Advanced, 970
    OmniView, 945
    PC-Mix, 969
    PC-MOS/386, 946
    TopView, 823
    VMiX, 901
    background execution, 603
    critical sections, 404
    OS/2, 623
    per-task data area, 623
    pipes, 609
    process priority, 607
    process blocking, 609
    process identifiers, 606
    shared memory, 605
    sleep, 606
    start/stop process, 604
    start/stop process, 604
    waiting for child, 607
    yielding CPU, 385
music;
    background, 1008
    FGDRIVER, 1008
MVSOUND.SYS, 472
    DMA channel, 472
    executable path, 473
    function table, 472
    installation check, 472
    IRQ channel, 472
    state table, 472
    status string, 472
    version check, 472
MX5 Extended FOSSIL;
    MNP level, 262
    MNP status, 262
    MNP control, 263
    status block, 262
    uninstall, 263
MX5 Extended FOSSIL;
    delay, 264
    MNP control, 263
    installation check, 263
    MNP control, 263
MZ executables, 533
Nabbit, 1144
Nanosoft, 887
Nanosoft, Inc.;
    CAPDOS, 1039
Nanosoft;
    see also MD386, 802
National Language Support;
    in MS-DOS, 637
    see also NLSFUNC, 637
National Language Support;
    in OS/2, 624
NB.SYS, 1166
    state, 1167
    state, 1166
NCACHE;
    flushing buffers, 1103
    rebooting, 1103
NDOS;
    API, 647
    installation check, 645
    prompt callout, 647
NDOS;
    see also 4DOS, 645
NDOSSTAK device, 523
NEC PC-9800, 80
    printer, 81
    get keystroke, 80
    keyboard, 80
    keyboard, 80
    keyboard, 80
    keyboard, 80
NEC, 80
    shift keys, 80
    printer, 81
    video, 80
    printer status, 81
NetBIOS;
    VNETBIOS driver, 814
Netroom, 683
Netroom;
    CACHECLK, 361
    DPMI.EXE, 792
    installation check, 783
    NETSWAP4, 793
    POSTNET, 783
    PRENET, 783
    SCRNCLK, 167
    see also RM386, 783
    switching to protected mode, 785
    undetermined functions, 783
    XLOAD, 792
NETSWAP4, 793
network;
    server function call, 567
new executables, 533
new executable format, 535
New-Ware, 1170
Newkey;
    installation check, 285
NewSpace, 329
NewSpace;
    debugging dump, 330
    disabling driver, 330
    enabling driver, 330
    installation check, 330

  - NewRes, 330
  - resident code segment, 330
  - undetermined functions, 330
- Nick Zaikin, Jr., 1146
- NJFRERAM, 1145
- NLSFUNC, 637
- NLSFUNC;
  - code pages, 638
  - code pages, 637
  - country information, 638
  - DR-DOS, 620
  - DR-DOS, 619
  - extended country information, 638
  - installation check, 637
- NMI, 25
- NMTx device, 223
- non-maskable interrupt, 25
- non-volatile memory;
  - Amstrad, 88
- Northgate, 148
- NorthNet Jetstream—see Jetstream, 300
- Norton Utilities;
  - EP.INI, 1102
  - FILESAVE.INI, 1102
  - rebooting, 1103
  - enable/disable, 1103
  - installation check, 1101
  - flushing buffers, 1103
  - enable/disable, 1102
  - DISKREET, 1102
- Norton Utilities, 1101
- Norton Guides, 1101
- Norton Utilities;
  - DISKMON.INI, 1102
- Norton Utilities, 1103
- Novell DOS;
  - undelete file, 615
- Novell DOS;
  - see also DR-DOS, 613
- Novell DOS, 613
  - purge deleted files, 615
- NOW!, 341
  - cache information, 341
  - cache information, 341
  - hit rate, 342
  - information record, 341
  - resident code segment, 341
- NRMTx device, 223
- Nu-Mega Technologies, Inc., 1050
- NUL device, 557
- NumLock, 1004
- NumLock, 1151
- Oak Technologies VGA, 155
- OEM INT 21h, 581
- OEM INT 21h, 581
- Olivetti;
  - *see also AT&T 6300*, 89
- OmniView, 945
  - background processes, 946
  - callouts, 945
- OmniView;
  - device information tables, 945
  - entry point, 945
  - exit notification, 946
  - process management, 945
- OmniView;
  - process management, 945
- OmniView;
  - save/restore state, 945
- OmniView;
  - save/restore state, 945
- OMS Development, 490
- OMTI controllers;
  - disk parameters, 217
- Ontrack Computer Systems, 215
- opcodes;
  - breakpoint, 22
  - invalid, 22
- open file limit, 606
- open files, 913
  - via FCB, 499
- opening files, 580
- opening file;
  - OS/2, 626
- operating system hooks;
  - device busy, 41
  - device close, 38
  - device open, 38
  - device post, 42
  - program termination, 38
  - SysReq activity, 39
  - SysReq routine, 34
  - SysReq routine, 34
- operating system indicator, 59
- OPTHELP, 1176
- OPTIMA;
  - application key, 200
  - specify mode, 199
  - window location, 200
  - window location, 200
  - zoom offset, 200
  - zoom interrupt, 199
  - zoom factor, 199
  - zooming, 199
- Oracle SQL, 1025
- Orchid ProDesigner IIs—see Tseng, 197
- OS/2;
  - Family API, 621
  - extended attributes, 627
  - compatibility box, 627
  - API ordinals, 625
  - creating directories, 626
  - DosENumAttrib, 627
  - DosOpen2, 626
  - DosMkDir2, 626
  - DOSKRNL, 624
  - DosQMaxEASize, 627
  - compatibility box, 627
  - Virtual DOS Machines, 624
  - WinOS2 call gate, 624
  - WinOS2, 623
  - Virtual DOS Machines, 626
  - JFT, 623
  - Virtual DOS Machines, 623
  - opening files, 626
  - Virtual DOS Machines, 622
  - National Language Support, 624
  - loading message, 624
  - LASTDRIVE, 623
  - relocated IRQs, 627
  - OS2KRNL, 624
  - Virtual DOS Machines, 622
  - SFT flags, 624
  - VDM OS/2 API support, 625
  - STARTDATA structure, 626
  - unload debugging information, 624
  - API request buffer, 705
  - calling real mode, 706
  - VMM functions, 706
  - segment manipulation, 707
  - segment creation, 706
  - CPU idle, 809
- OS2KRNL, 624
- OSARGS, 84
- OSARGS, 84
- OSASCI, 86
- OSBGET, 84
- OSBPUT, 84
- OSBYTE, 87
- OSCLI, 87
- OSFILE, 85
- OSFIND, 83
- OSGBPB, 83
- OSNEWL, 86
- OSRDCH, 85
- OSWORD, 86
- OSWORD;
  - control block, 86
- OSWRCH, 86
- output to file, 86
- output;
  - HP 95LX, 389, 427, 433
  - to serial port, 247, 251, 256-274
  - to screen, 257, 389, 427-433, 496, 601, 825-850, 995-996, 1094, 1053-1058
  - to HP 95LX communications channel, 421
  - to printer, 418
  - to standard output, 496
  - to AUX device, 926
  - to serial port, 948
- overflow, 22
- overlay manager, 1021
  - Turbo Debugger, 1054
- overlays, 532
- overscan color, 124
- P.R. Glassel Associates, Inc., 344
- P2 Enterprises, 279
- page fault, 24
- page description language, 301
- page tables, 698-728
- page tables;
  - QEMM-386, 765
- page tables, 728
- paging;
  - dempaging, 673
  - DPMI real-mode memory, 672
  - locking memory, 656
  - locking pages, 702
  - page replacement, 697
  - page attributes, 669
  - page size, 672
  - page attributes, 669
  - page table entries, 755
  - page tables, 777
  - page protection, 776
  - page tables, 790
  - page tables, 789
  - page tables, 909
  - paging candidates, 672
  - read-only, 789
  - read-only pages, 803
- Paint Tools, 1019
- palette registers, 123
- palette, 161
- palette registers, 188
- palette registers, 990
  - locking, 165
- panning display, 433
- Paradise VGA, 146
- parallel port;
  - number installed, 31
- parity error;
  - Z100, 97
- parity, 926
- PARKER, 1139
- parking disk, 214
- parking disk, 235
- partition record, 59
- passwords;
  - After Dark, 289

  After Dark, 289
  DR-DOS, 499, 503, 518, 614
  MDEBUG, 1047-1050
path separator, 590
PC-9800;
  *see NEC PC-9800*, 80
PC-Cache, 215
  huge disks, 215
PC Paint Plus;
  status, 318
  printer type, 318
  undetermined functions, 318
  paper size, 318
  page eject, 319
  unimplemented function, 319
  print spooler, 317
PC Magazine;
  COURIERS, 266
  CAPTURE, 281
  PCSpool—see PCSpool, 310
  TEXTCAP, 281
PC-Cache;
  enabling, 344
  disabling, 343
  flushing, 343
  Bernoulli drives, 359
  delayed writes, 343
  delayed writes, 343
PC-CACHE.SYS, 359
PC-Kwik programs, 360
  installation check, 359
  hit rate, 343
  flushing, 344
PC-Kwik—see Super PC-Kwik, 354
PC-Xview, 874
PC-MOS/386;
  security, 957
  selector aliases, 956
  remote printer redirection, 953
  remote printer flags, 953
  serial input, 948
  serial I/O, 946
  serial driver description, 948
  serial input, 949
  processor mode switches, 955
  print spooler, 960
  program name, 957
  protected mode, 955
  Record Lock Block, 955
  registering ports, 948
  Native Context Area, 955
  serial protocol options, 946
  serial protocol options, 946
  serial status, 949
  System Control Block, 950
  System Control Block, 950
  serial port status, 947
  serial port paramters, 958
  serial port break, 947
  serial port ownership, 949
  serial port parameters, 947
  serial number, 959
  System Control Block, 959
  control block access, 960
  control block access, 959
  Current Directory Block, 954
  device driver address, 959
  disabling port, 947
  control block access, 959
  clearing serial buffers, 947
PC-Mix, 969
PC Tools;
  CPTASK, 960
PC-MOS/386, 946
  addtask data structure, 958
  Block Device Block, 954
  error codes, 958
  event types, 955
  keyboard mode, 957
  IRQ reservation, 956
  linking serial drivers, 948
  memory allocation, 956
  memory allocation, 956
  IOCTL flags, 953
  installation check, 950
  extended directory information, 950
  event wait, 955
  extended directory information, 951
  flushing serial buffers, 949
  Global File Block, 954
  modem control register, 948
  System Memory Pool, 958
  TCB, 949
  TCBWAIT field, 955
  terminal driver, 959
  user name, 957
  undetermined functions, 949
  time slices, 957
  task size, 960
  task priority, 956
  Task Control Block, 951
  Task File Block, 953
  task ID, 956
  task partition, 957
  task management, 958
  version check, 950
PC-IPC, 1173
PC-IPC;
  commcodes, 1173
PC/370, 1177
PC/370, 1174
PC-IPC;
  error codes, 1174
PC Tools;
  DESKTOP, 1086-1100
  PCShell, 1095
  PCRUN, 1096
  DATAMON, 1099
  PCRUN, 1096
  DRIVEMAP, 1085
  DESKTOP, 1099
  DESKTOP, 1097
  DATAMON, 1090
PC Thuis Shell;
  memory copy, 1159
  memory compare, 1159
  keyboard input, 1159
  memory handle, 1160
  menus, 1157
  messages, 1158
  messages, 1158
  menus, 1159
  keyboard input, 1156
PC Thuis Shell;
  hotkeys, 1159
PC Thuis Power magazine, 1155
  attributes, 1158
  clearing screen, 1159
  file I/O, 1156
  editing, 1158
  requests, 1158
  screen output, 1155
  CPTASK, 1085

PCL-838;
  installation check, 1154
  commands, 1155
PCM driver;
  pause/resume, 471
  installation check, 469
  initialization, 470
  pause/resume, 471
  recording, 471
  version check, 469
  uninstall, 471
  start/stop audio, 471
  initialization parameters, 470
  DMA control, 471
  initialization parameters, 470
  initialization, 470
  DMA control, 470
  initialization, 470
  DMA buffer, 471
PCM.COM, 471
  board address, 474
  Control-G interception, 476
  DMA info structure, 475
  hotkey sounds, 476
  hotkey simulation, 476
  idleness, 476
  information, 477
  installation check, 474
  installation check, 469
  preloading sounds, 475
  random sounds, 476
  see also PCM driver, 469
  see also PCM driver, 474
  signature string, 474
  single-step queue, 475
  sound data, 475
  status, 474
  status, 474
  timer control, 477
PCMANAGE/DCOMPRES;
  disabling, 334
  installation check, 334
PCMCIA Socket Services;
  callbacks, 73
PCMCIA Socket Services;
  callbacks, 73
PCMCIA Socket Services, 75
  error detection, 75
  attached devices, 72
  attached devices, 73
  error detection, 76
  error detection, 76
  I/O, 75
  error detection, 75
  status, 73
  version check, 73
  I/O, 75
PCnet, 27
PCRUN, 1095
PCRUN;
  installation check, 1096
PCShell, 1096
  installation check, 1095
  PCRUN support, 1095
  popping up, 1096
PCSpool;
  aborting output, 312
  control block, 310
  flushing buffers, 311
  installation check, 312
  pausing printing, 311
  printer status, 311
  queue status, 312
  shutdown, 312
  skipping jobs, 312
PCW Weather Card;
  uninstall, 1161
  installation check, 1161

  data segment, 1160
PCX files, 1002
Pdisk, 218
PEL mask, 125
Personal Calendar, 985
Petr Horak, 1144
PGS1600.DEV;
  configuration information, 174
  configuration information, 175
Phantom of the Keyboard—see Phantom2, 279
Phantom2, 279
  function mask, 280
  hotkey, 279
  installation check, 279
  undetermined function, 280
  uninstall, 280
Phar Lap;
  .EXP file header, 543
  run-time parameters, 543
  .EXP file header, 543
  VMM, 697
  VMM error codes, 695
  selector structure, 692
  physical addresses, 691
  paging, 696
  program load parameter block, 694
  protected-mode interrupt vectors, 690
  paging, 695
  page tables, 699
  page replacement, 697
  page locking, 696
  page tables, 696
  real-mode addresses, 692
  real-mode call parameter block, 693
  segment manipulation, 695
  segment movement, 695
  swap file, 700
  VMM, 695
  system registers, 699
  swap space, 697
  VMM, 697
  resignalling interrupts, 690
  real-mode interrupt parameter block, 693
  real-mode callbacks, 692
  reset DOS extender, 689
  real-mode interrupt vectors, 690
  page faults, 697
  MS-DOS memory allocation, 700
  entry point, 702
  DOS data buffer, 695
  conventional memory usage, 697
  error codes, 691
  execute parameter block, 701
  executing programs, 701
  configuration information, 698
  calling real mode, 693
  286|DOS-Extender Lite, 702
  286|DOS-Extender Lite, 701
  alias descriptors, 694
  available memory, 696
  calling real mode, 693
  extended memory usage, 697
  files mapped into memory, 699
  memory mapping, 699
  memory mapping, 691
  memory error information, 695
  memory statistics, 696
  MS-DOS memory allocation, 700
  memory allocation, 695
  mapping structure, 699
  installation check, 701
  IF virtualization, 699
  IDT descriptor, 700
  installation check, 701
  IRQ mapping, 691
  linear addresses, 691
  LDT, 695
  VMM, 695
  segment manipulation, 690
  VMM, 697
Phoenix BIOS;
  CPU speed, 96
physical addresses, 650
physical memory, 696
PIPELINE, 273
pipes, 609
pixels;
  reading, 122
  setting, 121
  writing, 992
pointing device;
  hardware interrupt, 30
  installed, 31
  joystick, 39
  see also mouse, 437
  sampling rate, 438
  status, 439
  user handler, 439
  type, 438
  sampling rate, 438
  resolution, 438
  enabling/disabling, 437
  commands, 439
  error codes, 437
  initialization, 438
  resolution, 438
  resetting, 437
polling, 242
popping up;
  AMIS, 973
  XVIEW, 1168
  XVIEW, 1169
  XVIEW, 1169
  XVIEW, 1169
  ZIPKEY, 1084
  PCShell, 1096
  XVIEW, 1169
  MDEBUG, 1049
  MDEBUG, 1049
  XVIEW, 1168
  PCShell, 1095
POPUP, 978
POPUP.COM;
  info structure, 456
  installation check, 456
POPUP;
  installation check, 610
POS—see programmable option select, 47
POST;
  error log, 34
POSTNET, 783
Postscript, 301
Postscript conversion;
  PSPS, 302
power management;
  IBM Convertible, 37
  modem, 37
  system shutdown, 36
  HP 95LX, 94
  HP 95LX, 95
  Compaq SLT/286, 91
  Compaq, 94
  Compaq SLT/286, 90
  SOLLEX, 192
  SOLLEX, 192
  video, 192
  video, 193
  video, 192
  video, 192
  SOLLEX, 192
  SOLLEX, 193
  APM, 371
  disabling, 374
  events, 375
  power status, 377
  stand-by mode, 374
  statistics, 378
  power status, 375
  power state, 373
  events, 376
  idle detection, 377
  idleness, 373
  suspend, 374
POWER.EXE;
  event broadcasting, 376
  idle detection, 377
  idle time, 378
  installation check, 376
  polling frequency, 378
  power management setting, 377
  power status, 377
  power management setting, 377
  statistics, 378
  undetermined functions, 378
PrecisePoint;
  see Smooth Mouse Driver, 461
PRENET, 783
Priam EDVR.SYS, 214
Priam hard disk controller, 215
primary shell, 913
PRINDIR, 316
  capture file, 316
  hotkey, 316
  installation check, 317
  port selection, 316
print screen, 31
print spoolers;
  Disk Spool II, 305
  PC-MOS/386, 317
  cancel file, 315
  PRINT—see PRINT, 312
  PCSpool, 310
  submit file, 314
  cancel file, 314
  PC-MOS/386, 960
print screen, 1016
PRINT, 312
  background printing, 313
  cancel file, 315
  critical section, 313
  error codes, 314
  installation check, 314
  job status, 315
  job status, 315
  output device, 315
  submit file, 314
  submit packet, 314
  undetermined functions, 313
  undetermined functions, 314
printer redirectors;
  LPTx, 316
  LPTx, 316
  PRINDIR, 316
  PRINDIR, 316
  LPTx, 316
  LPTx, 316
printer;
  aborting output, 301
  capturing output, 302
  character conversion, 80

  - country codes, 79
  - flushing buffers, 311
  - hardware interrupt, 28
  - HP 95LX, 418
  - initialization, 81
  - Japanese characters, 79
  - Japanese characters, 79
  - NEC PC-9800, 81
  - output, 79, 81, 301, 418, 496
  - page eject, 319
  - paper movement, 300
  - paper size, 318
  - paper movement, 299
  - Postscript conversion, 301
  - status, 81
  - status, 295, 301, 311
  - Z100, 97
- printing graphics, 1032
- priority, 889-956
- Pro Audio Spectrum—see also Media Vision, PCM.COM,
- process ID, 546- 573
- process identifiers, 606
- process termination, 607
- process priority, 608
- process identifiers, 904
- process termination, 906
- process ID, 956
- process ID, 1020
- processes;
  - starting, 404
  - starting, 906
  - terminating, 404
- processor modes;
  - switching to protected mode, 41
  - shutdown, 22
  - returning to real mode, 22
  - detection, 653
  - switching, 665
- processor exceptions;
  - FlashTek, 703
- profilers, 1035
  - Microsoft Profiler, 1035
  - SYS_PROF, 1037
- program termination, 38
- program termination address, 597
- program termination, 932
- Program Segment Prefix—see PSP, 506
- programmable option select, 47
- Progressive Solutions, Inc., 929
- protected mode;
  - ABIOS, 34
  - returning from, 63
  - 16-bit, 372
  - 32-bit, 372
  - calling real-mode code, 664
  - calling real-mode code, 664-70
  - switching to, 757
  - switching from, 758
  - Cloaking, 784
  - executing programs, 727
  - calling real mode code, 720
  - switching from, 787
  - switching to, 785
  - executing in, 910
- PROTMAN, 642
- PrtSc key, 940
- PSP, 506
- PSP segment, 547, 573, 785, 1090
  - child PSP, 561
- PSPRINT, 319
  - page eject, 302
  - parameters, 302
  - parameters, 302
  - uninstall, 302
- PTxxx.COM, 1019
- PUSHDIR.COM, 1140
- PUSHF instruction, 674
- Q-PRO4, 1177
- QDPMI;
  - installation check, 1108
- QDPMI$$$ device, 1108
- QEMM-386, 365
- QEMM-386;
  - memory mapping, 757
  - memory types, 758
  - memory type map, 758
  - memory usage statistics, 762
  - memory allocation, 755
  - memory available, 756
  - memory flags, 765
  - memory access status, 762
  - Stealth, 764
  - Stealth ROM information, 768
  - switch to protected mode, 757
  - ST-DBL support, 767
  - protected-mode interface, 756
- QEMM-386;
  - page table entries, 755
  - MS Windows support, 763
  - page table entries, 755
  - page tables, 765
  - physical addresses, 757
  - memory usage statistics, 767
  - callbacks, 760
  - debugging registers, 756
  - DESQview hooks, 759
  - EMM Import data structure, 764
  - DESQview/X support, 766
  - CR0 register, 757
  - switch to V86 mode, 758
  - interrupt controller, 760
  - asynchronous disk accesses, 765
  - bugs, 754
  - entry point, 753
  - entry point, 753
  - installation check, 754
  - mapping context, 755
  - installation check, 753
  - I/O port access, 763
  - hardware interrupts, 764
  - entry point, 754
  - Hercules mode-change support, 760
  - highest address, 757
  - HIRAM chain, 758
  - memory access status, 762
  - version check, 755
  - VIDRAMEGA, 758
  - VGA virtualization, 760
  - VirtualHDIRQ, 765
- QEMM.COM, 758
- QEMM;
  - state, 754
  - VCPI protected-mode entry point, 649
- QEMM386$ device, 753
- QEXT.SYS, 883
- QMAPS, 365
- QMAPS, 804
- QMMXXXX0 device, 769
- QMR—see Quick Mouse Reset, 462
- QPMS, 777
- Quadtel, 804
- Qualitas Qcache, 354-357
  - installation check, 359
- Qualitas Protected Memory Services, 777
- Qualitas—see 386MAX, 769
- Quarterdeck Office Systems, 856
- Quarterdeck programs;
  - VIDRAM, 1107
  - QEMM-386, 1106
  - QRAM, 1106
  - QDPMI.SYS, 1108
- Quarterdeck RPCI;
  - installation check, 1105
  - high memory chain, 1106
  - DVDOS4GX.DVR, 1106
- Quarterdeck;
  - QEMM-386, 753
- queues, 906-931
- Quick Mouse Reset;
  - disabling, 463
  - resident code segment, 463
  - see also mouse, 462
  - hardware reset, 463
  - enabling, 463
  - resident code segment, 463
  - installation check, 462
- QuickCache II;
  - sector buffers, 345
  - sector locking, 348
  - resizing, 345
  - sector weighting, 350
  - sector locking, 348
  - sector weighting, 350
  - incremental flushing, 346
  - information, 346
  - installation check, 345
  - flushing, 350
  - undetermined functions, 349
  - uninstall, 345
  - sounds, 346
  - system info, 347
  - tracing, 348
  - single-sector transfers, 350
  - conventional memory reservation, 348
  - delayed writes, 346
  - buffered reads, 349
  - buffered reads, 349
  - access frequency, 346
  - buffer size, 345
  - buffered reads, 346
  - delayed writes, 349
  - disabling, 344
  - flush count, 346
  - flush interval, 345
  - flushing, 344
  - enabling, 348
- QuickCache II;
  - drive info, 347
  - disabling, 348
  - flushing, 344
  - dismount, 344
  - dismount, 345
- RAID, 1161
  - installation check, 1161
  - resident code segment, 1161
  - uninstall, 1161
- RainbowFAX, 1123
  - see also SFENGINE, 1122
- Rainer Schuetze, 334
- RAM disks;
  - COMBI-disk, 361
  - xDISK, 219
  - xDISK, 219
  - xDISK, 219
- RAMLIGHT, 1172
- range checking, 22
- Rational Systems -- see DOS/16M DOS/4G, 709
- RATSR, 1064
- RBcomm, 865
- reading files, 397

    via FCB, 502
real-time clock;
    1024Hz interrupt, 40
    hardware interrupt, 29
    HP 95LX, 29
real-time clock;
    time, 61
real-time clock;
    time, 60
    status registers, 88
    switching to, 63
    alarm, 61
    system powerup, 62
    alarm, 62
    switching to, 63
    date, 61
    calling from protected mode, 664-730
    switching to, 787
real-time clock, 1140
Realtek RTVGA, 141
reboot panel, 641
rebooting, 58
    EMS, 745
    Z100, 98
recalibrate, 203
record locking, 566
Red E Products, 1177
REDVIEW, 1144
reentrancy problems, 312
reference partition, 53
Register Interface Library, 155
registers;
    CR0, 650
    CR3, 652
    debugging, 651
    GDTR, 757
    GS, 713
    IDTR, 652
    IF, 674
    LDTR, 652
    machine status word, 757
    TR, 652
Reidar Gresseth, 291
relocated IRQs, 627
relocated IRQs, 873
relocated IRQs, 942
relocated IRQs, 1162
Relocated Screen Interface Specification, 823
Relocated Screen Interface Specification, 824
remote files, 524
remote control software;
    RATSR, 1064
remote control software;
    pcANYWHERE, 1060
remote control software;
    Carbon Copy Plus, 1059
    COMMUTE, 1059
RemoteAccess, 1064
renaming files, 400
    via FCB, 503
    via FCB, 561
request packet;
    Compaq AG1024.SYS, 176
    Compaq AG1024.SYS, 177
    RGDI, 177
    CPQ_MGES, 176
reset, 22
resident segment;
    4DOS, 645
    KEYB, 621
    DVDOS4GX, 876
    QEXT.SYS, 883
    DVDOS4GX, 876
    Headroom, 943
    Ross Wentworth's Turbo Pascal
    Kingswood TSRs, 980
    viruses, 1071
    VSWAP, 1167
    SitBack, 1165
    BACKTALK, 1085
    Turbo Lightning, 1171
    VSWAP, 1167
    RAID, 1161
    COMMUTE, 1060
    CPSCHED, 1086
    DESKTOP, 1097
    MDEBUG commdriver, 1047
resizing files, 520
resizing memory block, 700
resource management;
    printer, 417
    serial, 419
RESPLAY, 479
    initialization, 480
    installation check, 480
    playback, 479
    sample rate, 480
    sampling, 479
    uninstall, 480
retrace rate, 196
retrace rate, 196
retrace rate, 196
return codes, 607
Rex Conn, 645
REXX88PC, 1176
Rhode IslSoft Systems, Inc., 289
RI signal, 927
Rich Bono, 1057
Right HMan, 1177
Right HMan, 1143
RighTime, 1161
RighTime;
    enable/disable, 1162
    enable/disable, 1162
    enable/disable, 1162
RM386, 365
    adding upper memory, 784
    address map, 786
    Compaq computers, 788
    bugs, 688
    Cloaking, 683
    EMS handles, 788
    enabling, 789
    global flags, 788
    HMA allocation, 784
    I/O port trapping, 791
    I/O port trapping, 791
    installation check, 786
    linear address, 785
    memory protection, 789
    memory mapping, 790
    memory types, 786
    memory mapping, 789
    memory manager size, 786
    page tables, 790
    resident code segment, 786
    resident code segment, 786
    status, 790
    switching to real mode, 787
    undetermined functions, 792
    XMS handles, 784
    XMS extensions, 784
    XMS, 787
    XMS services, 792
RMDIR, 401
RMTx device, 223
Robert Curtis Davis, 977
Roger Cross, 339
ROM BASIC, 35
ROM configuration table, 43
ROM BIOS;
    tracing callout, 47
ROM scan, 581
ROM modules, 535
ROM programs, 580
ROM programs, 580
ROM Windows;
    see also MS Windows, 809
ROM;
    COMMAND.COM, 644
    detection by MS Windows, 809
root shell, 913
Ross Wentworth, 978
Ross Neilson Wentworth, 1161
Ross M. Greenberg, 1076
RPL code, 641
RSE Inc., 1144
RSIS, 823
RSIS, 824
RTC;
    *see real-time clock*, 88
RTM, 725
    installation check, 725
    undetermined functions, 725
    undetermined functions, 726
RTS, 927
runtime support, 1019
S100 bus;
    Z100, 98
    Z100, 97
    Z100, 98
    vectored lines, 98
S3;
    video modes, 186
Sample Entry, 3
SBFM driver;
    version check, 467
    user trap, 469
    start/stop music, 468
SBFM driver;
    pause/resume music, 469
    reset, 469
    pause/resume music, 469
    music status, 468
    driver clock rate, 468
    instrument table, 468
    start/stop music, 468
    system clock rate, 468
    system-exclusive commands, 469
    transposing music, 468
    translation, 37
scan lines;
    per screen, 129
    length, 187
ScanBoot, 1074
scatter/gather, 367
scatter/gather, 368
SCOUT, 1170
SCOUT;
    installation check, 1170
screen blanking;
    Compaq, 90
screen height, 129
screen blanking;
    Compaq, 90
screen saver;
    ATI Wonder SMS.COM, 166
screen saver;
    Explosiv, 292
    Burnout Plus, 293

  Blank-It, 291
  FASTBUFF, 280
  InnerMission, 294
  KEYBUI, 280
  SCRNSAV2, 294
  Blank-It, 290
  AD-DOS, 287
  VGAsave, 294
screen readers;
  AutoBraille, 486
  Tinytalk Personal, 490
  ASAP, 485
  AccessDOS, 485
screen magnifiers;
  MAGic, 488
Screen saver;
  ADAPT, 167
Screen Thief;
  uninstall, 1162
  relocated IRQs, 1162
SCREENR;
  installation check, 165
  mode locking, 165
  palette locking, 165
SCRNCLK, 167
SCRNSAV2, 294
scrolling;
  AX PC video, 78
SCROLLit, 1143
ScrollLock, 1004
SCROLOCK;
  enable/disable, 165
SCROLOCK;
  installation check, 165
SCSI commands, 233
SCSI, 221
SCSI commands, 240
SCSI bus;
  reset, 232
SCSI commands, 232
SCSI commands;
  commformat, 232
SCSI Request Block, 221
SCSI commands;
  SDLP, 243
SCSI.SYS, 243
SCSI.SYS, 244
SCSI;
  address, 241
  CD-ROM redirector, 245
  CD-ROM redirector, 244
  controller information, 237
  defect tables, 235
  device types, 237
  device information, 239
  device capabilities, 237
  disk information block, 239
  disk information block, 239
  DMA channel, 241
  IBM SCSI interface, 245
  identify devices, 243
  inquiry information block, 236
  medium capacity, 238
  mode select, 234
  mode select, 234
  reset, 240
  see also Common Access Method, 224
  see also SDLP, 242
  see also SCSILink, 240
  see also Future Domain, 234
SCSI_CAM device, 224
SCSIDISK device, 244
SCSILink, 240
  address, 241
  commblock, 240
  connect, 240
  data transfer, 241
  data transfer, 241
  error string, 241
  parameter block, 240
  polling, 242
  reset, 240
  send command, 240
  shutdown, 241
  timeout, 240
  timeout selector, 241
SCSIMGR$ device, 221
SCSITAPE device, 243
SCSITAPE device;
  control block, 244
SD62.COM, 1176
SDA, 568
SDLP, 242
  commands, 242
  generic SCSI command, 243
  issue command, 242
  WD7000, 243
Seagate disk controller, 214
Seagate ST-01, 243
Seagate disk controller;
  parking heads, 214
Seagate disk controller, 214
search path, 631
SeaSide Software, 1170
Seaware, 1174
SEGDEBUG interface, 1036
segment wrap, 24
segment access rights, 657
segment base address, 707
segment access rights, 694
selector validity, 729
selectors, 812
  aliases, 956
  allocating, 683
  allocating, 683
  increment between, 656
selectors;
  locking, 656
self-extracting archives, 535
semaphores, 894
serial I/O;
  hardware interrupts, 28
  hardware interrupts, 27
  number installed, 31
  Z100, 97
  Z100, 97
  status, 254
  word length, 249
  status, 250
  speed, 249
  character output, 258
  speed, 250
  status, 248
  character input, 247
  break, 258
  character output, 247
  reading characters, 254
  character output, 251
  DTR control, 255
  initialization, 254
  input availability, 251
  line status, 248
  flushing buffers, 251
  readiness to transmit, 252
  flushing buffers, 255
  modem control register, 249
  flow control, 256
  output character, 256
  modem status, 248
  parameters, 247
  parity, 249
  protocol, 250
  port control, 249
  input availability, 274
  modem control register, 260
  initialization, 271
  port activity check, 267
  parity, 260
  port control, 259
  initialization, 264
  flushing buffers, 270
  status, 269
  timeout, 270
  status, 268
  speed, 268
  recapturing port, 269
  shutdown, 268
  port control, 259
  word length, 260
  character input, 265
  aborting output, 268
  break, 268
  character input, 269
  configuration, 269
  DTR control, 264
  flushing buffers, 267
  configuration, 267
serial mouse;
  mouse port, 460
  report rate, 460
  emulation type, 460
  resetting, 459
  see also mouse, 459
serial port;
  modem status, 896
  line status, 897
  output, 896
  status, 898
  line status, 897
  input, 898
  buffers, 898
  parameters, 958
  PC-MOS/386, 946
  parameters, 947
  status, 947
serialization;
  DPMI, 677
serialization;
  DPMI, 677
server function call, 567
SET-ECHO.EXE, 469
SETDRVER, 1164
SETDRVER, 1163
SETDRVER;
  TaskMAX, 1164
SETDRVER;
  DOS version, 1163
  enable/disable, 1163
  installation check, 1163
  reset, 1163
SETDRVER;
  version check, 1163
SFENGINE, 1123
  see also RainboxFAX, 1122
SFT, 554
  in OS/2, 623
  OS/2 flags word, 624
shadow memory, 767
shadow RAM, 774
SHARE hooks, 551
SHARE, 638
SHARE.EXE, 551

SHARE.EXE;
  Record Lock Record, 553
  sharing record, 553
SHARE;
  DR-DOS, 617
  closing files, 639
  file sharing checks, 640
  file sharing checks, 640
  installation check, 640
  open file list, 639
  record locking, 566
  status, 617
  undetermined function, 640
shared memory;
  European MS-DOS, 605
  DRMDOS, 923
sharing retry count, 525
sharing interrupts, 971
SHELL=, 643
SHELL2E;
  uninstall, 645
SHELLB;
  COMMAND.COM interface, 933
  COMMAND.COM interface, 934
  installatino check, 933
  SHELLC interface, 933
  transient-to-TSR interface, 934
shells;
  VMiX, 907
shift keys;
  NEC PC-9800, 80
shift flags, 56
shift flags, 57
Shift-JIS, 79
Shift-JIS, 79
Show Partner F/X;
  start presentation, 165
Show Partner F/X;
  uninstall, 166
shutdown mode, 22
SI3.SYS, 1176
Siemens, 603
Sierra SC1148x DAC, 198
signals;
  European MS-DOS, 607
  European MS-DOS, 608
signature strings;
  Genius Mouse, 452
  Logitech mouse driver, 440
  PCM.COM, 474
  PCM driver, 469
  Logitech mouse driver, 457
  IBMSND driver, 480
  Mouse Systems POPUP.COM, 456
signature strings;
  Quick Mouse Reset, 462
signature strings;
  Tinytalk Personal, 490
signature strings;
  MIDI driver, 482
signature strings;
  MAGic, 488
  CodeRunneR, 488
  Ad Lib SOUND.COM, 466
  SoundBlaster speech driver, 467
  Z-Nix mouse driver, 464
  compressed executables, 534
  STARLITE MAC driver, 526
  self-extracting archives, 535
  compressed executables, 534
  self-extracting archives, 535
  MEMLIMIT, 782
  QEMM-386, 754
  EMM386.EXE, 799
  DESQview/X .DVP, 859
signature strings;
  TurboPower TSRs, 978
  BMB Compuscience Canada Utilities,
  AMIS, 972
  RainbowFAX, 1122
  PUSHDIR.COM, 1140
  Word Perfect Third Party Interface,
  ZIPKEY, 1081
  Nabbit, 1144
  StackMan, 1165
  FN32, 1144
  ASCII, 1146
  BACKTALK, 1085
  CUCKOO, 1145
  HyperPAD, 1177
  FaxBIOS, 1111
  executable files, 535
  Codeview, 537
  CEMM, 795
single-step interrupt, 21
single-stepping, 756
SitBack, 1165
  DTA, 1165
  installation check, 1164
  resident data segment, 1165
sleep, 907
SLR Systems, 1176
SLT/286;
  *see Compaq SLT/286*, 90
Small Computer Systems Interface—see SCSI, 221
Smart Prompt, 363
SMARTAAR device, 350
SMARTAAR device, 351
SmartDrive, 350
SmartDrive;
  Windows, 353
  cache locking, 351
  cache size, 353
  cacheability check, 353
  control block, 351
  delayed writes, 353
  device driver header, 353
  disabling, 351
  double buffering, 353
  error popup, 354
  flushing, 352
  hit rate, 352
  installation check, 352
  resetting cache, 352
  resizing, 351
  status, 353
  status record, 351
  status, 350
  undetermined function, 354
  DoubleSpace, 325
  DoubleSpace, 326
SMARTPMT, 363
SMD—see Smooth Mouse Driver, 461
Smooth Mouse Driver;
  enabling, 461
  disabling, 461
  installation check, 461
  future expansion, 462
  information, 461
  sacrifice character map, 462
  program information, 462
  primary/secondary bitmap, 461
  see also mouse, 461
snake into two columns, 691
SND, 478
  background music, 479
  busy flag, 478
  installation check, 478
  music string, 479
  pause, 478
  start/stop sound, 479
  start/stop sound, 479
Soft-ICE;
  display string, 1050
  execute command, 1050
  error codes, 1051
  breakpoints, 1051
SoftCom programs, 985
SoftLogic Systems, 1172
software interrupt, 2
Software Carousel;
  program status, 963
  program termination, 963
  processing commands, 964
  partitions, 965
  selecting task, 963
  rebooting, 964
  task switching, 964
  installation check, 962
  partition name, 963
  partition size, 964
  partitions, 963
  memory size, 965
  callback, 965
  callback, 965
  DOS re-entrance, 965
  error codes, 963
Software Concepts Design, 1076
SOLLEX SVGA;
  adapter control, 189
  adapter control, 190
  adapter control, 190
  video support control, 192
  display control, 191
  display control, 190
SOLLEX;
  adapter request, 190
  attached displays, 191
  display output setting, 190
  display type, 190
  information, 188
  power management, 192
  video registers, 193
sound boards;
  Ad Lib, 465
  Pro Audio Spectrum, 474
  SoundBlaster, 467
  Pro Audio Spectrum, 471
  Pro Audio Spectrum, 469
  address, 474
sound;
  APL, 1016
  background sounds, 473-481
  background, 1007
  background, 1008
  FGDRIVER, 1007
  FGDRIVER, 1008
  IBMSND driver, 481
  multiplexor, 477
  music string, 479
  note buffer, 474
  number of voices, 473
  pause/resume, 471
  playing notes, 282
  playing, 481
  preloading, 475
  rest, 478
  sampling, 479
  sampling, 481
  start/stop, 479
  TopView, 826

  - transposing music, 468
- SoundBlaster, 467
  - echo value, 469
  - SBFM driver—see SBFM driver, 467
  - SET-ECHO.EXE, 469
  - speech driver, 467
  - speech driver entry point, 467
- sounds;
  - beep, 408
  - disabling, 409
  - HP 95LX, 409
  - thud, 408
- spawning processes, 405
- speech, 483
- SPEECH.COM, 483
  - convert text, 483
- SPEEDSCR.COM, 174
- SpeedStar HiColor VGA—see also Tseng, 197
- Sperry PC, 143
- SQL Base, 1024
- SQL, 1025
- SSTORDRV.SYS, 332
- ST-DBL, 767
- Stac Electronics, 331
- stack fault, 24
- Stacker, 331
- Stacker;
  - boot record, 332
  - device driver address, 331
  - device driver address, 331
  - device driver, 332
  - driver address buffer, 332
  - stacker buffer, 331
  - STACVOL file, 331
  - volume number, 331
- STACKEY, 174
- STACKEY;
  - installation check, 282
- StackMan, 1165
  - installation check, 1165
  - stack management, 1166
- STACKS command, 27
- STACKS, 806
  - 386MAX, 778
- STACKVOL file, 331
- standard input—see STDIN, 498
- standard output—see STDOUT, 496
- standard input—see STDIN, 496
- standard error, 1144
- STARLITE architecture, 627
- static functionality table, 134
- status flags;
  - Tatung, 143
- status flags;
  - Taxan, 143
- status flags;
  - ATI, 143
- STDAUX, 496
- STDIN, 496
- STDOUT, 496
- STDPRN, 496
- Stealth, 764
- StopPrg;
  - installation check, 1055
- string output;
  - to screen, 131
- string length, 591
- string length, 592
- STSC APL*Plus/PC—see APL, 1013
- Stuart Russell, 283
- StuffIt, 284
- SUBST, 555
  - OS/2, 622
- Sunny Hill Software, 945
- Super Switch, 166
- Super PC-Kwik;
  - callouts, 359
  - disabling, 358
  - undetermined functions, 355
  - enabling, 358
  - flushing, 357
  - resident code segment, 357
  - program termination, 358
  - program load, 358
  - installation check, 359
- SuperBIOS;
  - Victor 9000, 96
- SuperBIOS—see Western Digital Super BIOS, 214
- SuperStor PRO;
  - 2XON, 332
  - SSTORDRV.SYS, 332
- SuperStor, 332
  - 2XON, 332
- SuperStor;
  - see also 2XON, 332
- SuperVGA, 183
  - mouse, 451
  - ATI, 100
  - SOLLEX information, 188
  - VESA information, 183
  - video modes, 186
- suspend/resume;
  - Compaq SLT/286, 90
  - HP 95LX, 95
  - IBM Convertible, 63
- suspend/resume, 376
- Swap Utilities, 977
- SWAPDT, 977
- swapping;
  - SWELL, 965
- SWBIOS, 215
- SWELL;
  - enabling/disabling, 965
  - information, 966
  - information buffer, 966
  - installation check, 965
  - uninstall, 966
  - verbose mode, 966
  - undetermined functions, 967
- Switch-It;
  - undetermined functions, 967
- Switch-It;
  - undetermined functions, 967
- Switch-It;
  - undetermined functions, 967
- SWITCHAR, 514
- SWITCHES=/W, 559
- Sybase DBLIBRARY, 1025
- Sybase DBLIBRARY, 1025
- Symantec, 1101
- SYNC, 568
- synchronization, 856
- SYS_PROF, 1037
- SYS_PROF;
  - enabling/disabling, 1037
  - profiling table, 1037
  - profiling table, 1037
  - resetting, 1037
  - status, 1037
- SysReq, 34
- system profile;
  - IBM Convertible, 36
- system timer, 25
- system model codes;
  - Dell, 46
- system model codes, 44
- system configuration information, 42
- system timer;
  - clock tick, 63
  - time, 60
- system resume vector, 63
- system clock;
  - SBFM driver, 468
- system profilers—see profilers, 1035
- system time, 1161
- System Enhancement Associates, 281
- System FOSSIL, 262
- System File Table, 553
  - reference count, 590
- System Enhancement Associates, 1140
- Systempro;
  - *see Compaq*, 90
- SYSVARS, 547
- T2PS;
  - parameters, 305
  - settings, 305
  - uninstall, 305
- TAME, 1141
- Tandon memory mapper, 803
- Tandy, 96
- Tandy 2000, 152
- Tandy Schoolmate Plus, 1176
- Tandy;
  - digital sound, 478
  - hard disk, 27
  - RAM refresh, 28
  - sound, 477
  - sound multiplexor, 477
- tape backup, 223
- TARGA.DEV, 230
  - diagnostics, 232
  - DMA channel, 231
  - early return mode, 231
  - I/O port, 231
  - reset, 232
  - SCSI command, 233
  - SCSI device number, 231
- task gate, 23
- task state segment;
  - invalid, 23
  - busy bit, 23
- task switchers;
  - TaskMAX, 911
  - Back & Forth, 929
- task termination, 889
- task switchers, 929
- task switchers;
  - DOSSHELL, 933
  - DOS Task Switcher, 935
  - Headroom, 943
  - CPTASK, 960
  - SWELL, 965
  - Switch-It, 966
  - Software Carousel, 962
- Task Switcher;
  - switcher ID, 938
  - startup info structure, 939
  - notification function, 936
  - version structure, 938
  - switcher ID, 938
  - callout chain, 935
  - callback info structure, 935
  - instance data, 939
  - API information, 938
  - callouts, 935
  - instance data, 939
  - callouts, 939
  - installation check, 937
  - entry point, 937
- TaskMAX, 911

TaskMAX;
  EMS usage, 911
  primary comminterpreter, 913
  EMS usage, 911
  cut paste, 915
  hotkeys, 912
  installation check, 911
  name table, 911
  open files, 913
  paste buffer, 915
  status, 911
  swap space, 915
  task manager, 915
  task menu, 911
  task name, 913
  task parameter block, 912
  task creation, 912
  task destruction, 913
  task index, 913
TaskMAX;
  task selection, 912
  task selection, 912
  version check, 911
Tatung video, 142
Taxan video, 142
TBSCANX, 1078
  state, 1078
  status, 1078
  virus scan, 1079
  virus scan, 1079
TDHDEBUG device, 1053
TDX;
  see also Turbo Debugger, 1054
TeleReplica;
  installation check, 1065
teletype output, 122
TEMPLEXX, 283
temporary files, 566
terminal emulation, 1151
terminating programs, 544
  RighTime, 1162
TesSeRact;
  hotkeys, 984
  installation check, 982
  uninstall, 982
  user parameter block, 984
testing;
  power-on self-test, 34
Texas Instruments PC, 179
TEXTCAP;
  dump screen, 281
  installation check, 281
  uninstall, 281
  input, 274
  input available, 274
  installation check, 274
  output, 274
  version, 274
The Software Bottling Company, 174
The Last Byte—see LASTBYTE.SYS,
    HIGHUMM.SYS, 800
The Software Link, Inc., 946
THELP, 983
ThesPlus, 985
threads, 889
threads, 889
ThunderByte, 1078
TI Professional PC, 216
TIGA Communications Driver;
  uninstall, 273
  installation check, 273
  entry points, 273
time, 509
timer tick, 63
timers, 833
  PCM.COM, 477
timestamp, 412
Tinytalk Personal, 490
  configuration info, 490
TKERNEL, 726
  calling real mode code, 730
  copying memory, 728
  descriptors, 727
  entry point, 726
  executing programs, 727
  installation check, 726
  memory management, 732
  page tables, 728
TKERNEL;
  page tables, 728
  parameter block, 730
  physical address, 729
  response buffer, 726
  uninstall, 732
  version check, 726
TMED, 1139
TopView;
  fill-in fields, 846
Tom Kihlken, 281
Top memory, 767
TOPS network, 27
TOPS network, 27
TopSPEED Modula-2, 1025
TopView, 823
  .PIF file format, 857
  ADDR message, 831
  APPEND, 630
  APPLY message, 833
  asynchronous notification, 846
  AT message, 854
  CLOSE message, 852
  CPU idle, 824
  critical sections, 827
  critical sections, 827
  cursor position, 835
  data streams, 842
  DIR message, 831
  EOF message, 853
  ERASE message, 852
  error popup, 828
  field mode, 850
  FREE message, 831
  GETSCALE message, 855
  HANDLE message, 830
  input from objects, 832
  keyboard filter function, 855
  keyboard objects, 850
  LEN message, 848
  LOCK message, 856
  logical attributes, 846
  mailbox messages, 831
  mailboxes, 830
  Manager streams, 843
  memory allocation, 824
  message interface, 830
  NEW message, 830
  notification messages, 847
  object destruction, 831
  object handles, 825
  object cursor, 854
  object length, 848
  object size, 848
  object bits, 849
  object bits, 851
  object creation, 830
  ObjectQ, 851
  objects, 825
  OPEN message, 851
  output to window, 834
  panel files, 831
  panels, 833
  pointer objects, 855
  positioning window, 828
  Query stream, 842
  READ message, 832
  reading objects, 854
  READN message, 854
  REDRAW message, 855
  screen output, 850
  screen updates, 824
  screen contents, 832
  screen output, 825
  see also DESQview, 823
  SETESC message, 855
  SETICON message, 855
  SETSCALE message, 854
  shadow screen, 823
  SIZEOF message, 848
  sound, 826
  STATUS message, 853
  SUBFROM message, 850
  suspending tasks, 827
  task creation, 830
  timers, 833
  user streams, 855
  version check, 828
  window locations, 826
  window movement, 843
  Window stream, 835
  WRITE message, 833
  writing to objects, 833
  writing to window, 834
Toshiba T1000 ROM MS-DOS, 581
touch screen, 181
TR register, 652
TRAP, 1055
Trident, 197
Trident Video BIOS, 197
  BIOS information, 197
  memory size, 197
TRUENAME, 572
truncating file, 397
truncating files, 520
Trusted Access, 977
  state, 1167
  state, 1166
Tseng, 197
  DAC type, 198
  ET-3000 chipset, 198
  video modes, 197
Tseng;
  video modes, 198
  zooming display, 198
TSR managers;
  Headroom, 943
TSR libraries;
  TesSeRact, 977
  TesSeRact, 982
  Ross Wentworth's Turbo Pascal
      POPUP,
  TSRBONES, 977
  Kingswood TSR interface, 980
  TurboPower, 977
  MicroHelp Stay-Res Plus, 985
  Microsoft TSR Specification, 977
  MicroHelp Stay-Res, 985
  CappaCom, 985
  BMB Compuscience Canada, 978
  CiriSOFT, 979
TSR managers;
  MIN-MEM, 1144
TSRBONES, 977

- TSRCOMM;
  - flushing buffers, 270
  - initialization, 270
  - installation check, 269
  - status, 271
  - timeout, 270
  - uninstall, 271
- TSRs;
  - MS Windows, 808
  - libraries, 977
- TSS, 906
  - Cloaking, 687
- TSS—see task state segment, 23
- Turbo Debugger, 24
- Turbo Pascal, 715
- Turbo Debugger;
  - COMTROL Hostess, 1040
- Turbo Debugger;
  - status, 1052
  - overlay manager, 1054
  - status, 1052
  - hardware breakpoints, 1053
- Turbo Lightning, 1171
- Turbo Debugger;
  - TDX, 1054
  - breakpoints, 1053
  - addressing modes, 1053
  - hardware breakpoint commands, 1053
  - breakpoints, 1053
  - configuration bits, 1052
  - error codes, 1052
- TurboPower TSRs, 978
- TurboPower Software, 1143
- typematic, 54
- Udo Chrosziel, 1055
- UILIB;
  - installation check, 1025
- UILIB;
- UltraVision, 160
  - cursor type, 162
  - extensions, 160
  - font names, 161
  - fonts, 163
  - hardware underlines, 163
  - installation check, 160
  - palette locking, 161
  - palette, 161
  - status, 160
  - text mode , 163
  - underline status , 162
- UMB, 549
- UMB memory chain, 565
  - adding blocks, 784
  - reallocating, 784
- underlined characters, 162
- uninstall;
  - PCW Weather Card, 1161
  - Show Partner F/X, 166
- uninstalls;
  - 2XON, 332
  - Alternate Multiplex Interrupt Specification,
  - AMIS, 973
  - BACKTALK, 1085
  - BitFax Scheduler, 1110
  - CAS Manager, 1138
  - CD-ROM redirector, 245
  - CiriSOFT TSR interface, 979
  - CPSCHED, 1086
  - CUBIT, 334
  - DBOS DOS extender, 734
  - DESKTOP, 1098
  - DESKTOP, 1095
  - DIET, 323
  - DOS/4GW, 710
  - DPMILOAD, 722
  - Emulaser ELTSR, 303
  - Explosiv, 292
  - F-POPUP, 1077
  - F-XCHK, 1076
  - F-DLOCK, 1077
  - F-LOCK, 1076
  - FAST!, 339
  - FastJuice, 1170
  - Jot-It!, 1152
  - Kingswood TSRs, 981
  - LapLink Quick Connect, 272
  - Logitech mouse driver, 459
  - MicroHelp Stay-Res, 985
  - Microsoft SQL Server, 1025
  - MX5 Extended FOSSIL, 263
  - OS/x86, 705
  - pcANYWHERE, 1063
  - PCL-838, 1154
  - PCM driver, 471
  - Phantom2, 280
  - PSPS, 302
  - QuickCache II, 345
  - RAID, 1161
  - RainboxFAX, 1122
  - RESPLAY, 480
  - Screen Thief, 1162
  - SFENGINE, 1122
  - SHELL2E, 645
  - SWELL.EXE, 966
  - Sybase DBLIBRARY, 1025
  - T2PS, 305
  - TesSeRact, 982
  - TEXTCAP, 281
  - TIGA Communications Driver, 273
  - TKERNEL, 732
  - TSRCOMM, 271
  - VSAFE, 1075
  - VWATCH, 1075
  - WHOA!, 1168
  - ZyXEL ZFAX, 1124
- Universal Data Exchange, 332
- University of Salford, 734
- Unix-compatibles, 969
- UNLINK, 520
- upper memory block, 549
- uppercase table, 575
- use transfer record described below), 557
- user-specified fonts, 126
- user interface;
  - Atari Portfolio, 89
- user-definable characters, 77
- V86 monitors, 24
- VADMAD virtual device API, 811
- Vanderaart Text Windows;
  - screen output, 1155
  - window management, 1155
  - screen output, 1155
  - keyboard input, 1156
  - attributes, 1156
  - editing, 1157
- VCPI, 649
  - available memory, 650
  - CR0 register, 650
  - debug registers, 651
  - installation check, 649
  - IRQ mapping, 651
  - maximum memory address, 650
  - memory allocation, 650
  - memory mapping, 650
  - protected-mode interface, 649
  - system register values, 652
- VDD—see VxD, 807
- VDEFEND, 1077
- VDEFEND, 1076
- VDISK;
  - header block, 58
- VDS, 365
  - address translation, 369
  - calls from Cloaked applications, 687
  - data transfer, 369
  - data transfer, 369
  - DMA descriptor structure, 366
  - DMA descriptor structure, 366
  - DMA buffers, 369
  - DMA descriptor structure, 366
  - DMA buffers, 369
  - DMA buffer allocation, 368
  - DMA buffer allocation, 368
  - error codes, 366
  - flags, 367
  - installation check, 365
  - locking, 368
  - product numbers, 365
  - scatter/gather locks, 368
- VEDIT—see VSWAP, 1167
- Vega VGA, 150
- verify flag, 510
- version checks;
  - Amstrad ROS, 89
  - PCMCIA Socket Services, 73
  - Cirrus Logic, 194
  - Cirrus Logic, 194
  - SCREENR, 165
  - ADAPT, 167
  - UltraVision, 160
  - Video FOSSIL, 168
  - MX5 Extended FOSSIL, 263
  - TFPCX, 274
  - ASSOC, 279
  - Virtual DMA Specification, 365
  - Super PC-Kwik, 359
  - APM, 371
  - SmartDrive, 352
  - PC-CACHE.SYS, 359
  - FAST!, 338
- version checks;
  - mouse driver, 447
  - PCM driver, 469
  - SBFM driver, 467
  - Logitech mouse driver, 460
  - Smooth Mouse Driver, 461
  - Logitech mouse driver, 457
  - MS-DOS, 510
  - MS-DOS, 513
  - Concurrent DOS, 615
  - APPEND, 630
  - DR-DOS, 616
  - 4DOS, 645
  - APPEND, 631
  - DPMI, 666
  - VCPI, 649
  - QEMM-386, 755
  - EMS, 739
  - TKERNEL, 726
  - DPMILOAD, 715
  - TopView, 828
  - MS Windows, 808
  - VTD driver, 811
  - WINOLDAP, 817
  - memory managers, 797
  - EGA.SYS, 817
  - DVSIXDI, 884
  - DESQview/X, 878
  - TaskMAX, 911
  - VMiX, 901
  - Back & Forth, 929

  - DESQview, 873
  - PC-MOS/386, 950
  - SWELL.EXE, 965
  - Image Processing Interface, 1031
  - SQL Base, 1024
  - BitFax Scheduler, 1109
  - ZIPKEY, 1081
  - SETDRVER, 1163
  - TBSCANX, 1078
  - DESKTOP, 1090
  - Futurus Team, 1143
  - pcANYWHERE, 1063
- vertical retrace interrupt, 27
- Vertisoft Systems, Inc., 304
- VESA Local Bus, 67
- VESA SuperVGA BIOS Extensions, 183
  - determining video mode, 186
  - disabling, 188
  - display start, 187
  - information, 183
  - installation check, 183
  - scan line length, 187
  - setting video mode, 185
  - video mode information, 184
  - video memory model, 185
  - video modes, 185
  - video memory control, 187
  - video modes, 186
  - video state, 186
- VFOSSIL— see Video FOSSIL, 168
  - ROM BIOS, 138
  - fonts, 138
  - palette, 130
  - vertical retrace, 27
  - video modes, 138
  - virtualization, 760
- VGAPAL.COM;
  - installation check, 174
- VGAsave, 294
- Victor, 96
- Victor 9000 SuperBIOS, 96
- VIDCLOCK.COM, 1143
- video state, 135
- video memory, 78
- video refresh, 131
- video modes;
  - retrieving, 122
  - setting, 99
  - information, 169
  - locking, 165
  - information about, 184
  - parameters, 137
  - information, 152
  - superimpose, 143
  - SuperVGA, 138
  - setting, 163
  - determining, 148
  - setting, 139
- video memory control, 187
- video modes, 185
- video mode;
  - Compaq master mode , 153
- video controller type, 194
- video state;
  - VESA, 186
  - UltraVision, 162
  - 24-bit, 201
  - ATI VGA Wonder, 143
  - supported, 151
- video mode set, 147
- video memory, 150
- video mode;
  - type, 151
  - true color, 201
  - setting, 391
  - determining, 391
  - setting, 448
  - determining, 987
  - FastGraph/Light, 987
- Video Electronics Standards Association, 183
- Video FOSSIL;
  - ANSI sequences, 170
  - application functions, 168
  - initialization, 168
  - information, 168
  - input, 172
  - output, 172
  - information, 168
  - error codes, 170
  - video mode information, 169
  - cursor shape, 169
  - output, 172
  - scrolling, 171
  - video configuration, 169
  - uninstall, 174
  - shutdown, 173
  - output, 172
- video;
  - 8514/A, 181
  - active page, 119
  - Amstrad, 88
  - GO32, 733
  - MS Windows, 816
  - attributes, 132
  - autoswitch mode, 150
  - BIOS information, 197
  - blinking, 123
  - bus width, 195
  - C&T emulation, 145
  - character attributes, 120
  - character generation, 128
  - character table, 138
  - character output, 389
  - clock rates, 189
  - color mix registers , 154
  - colors, 121
  - configuration, 150
  - country codes, 77
  - cursor emulation, 130
  - cursor position, 257
  - cursor shape, 162
  - cursor position, 391
  - DAC, 136
  - DAC registers, 124
  - DAC, 198
  - DAC registers, 188
  - determining mode, 122
  - disabling, 130
  - display memory size, 194
  - display memory address, 187
  - display disable , 154
  - display type, 190
  - display drivers, 163
  - display enable , 154
  - display memory address, 619
  - DMQS buffer, 137
  - DMQS data, 137
  - DMQS data length, 136
  - double-dot characters, 142
  - EGA information, 129
  - emulation , 155
  - emulation status, 150
  - emulation type, 144
  - enhanced features, 142
  - environment , 153
  - extended control register, 148
  - external monitor , 153
  - FastGraph/Light, 987
  - feature bits, 151
  - functionality information, 133
  - graphics bitmap, 94
  - graphics, 121
  - graphics characters, 141
  - graphics font, 138
  - gray-scale summing, 126
  - gray scale , 154
  - HDILOAD, 181
  - HiColor, 198
  - HP installation check, 147
  - HP monitor type code, 147
  - INCRAM address, 152
  - line width, 187
  - memory address, 150
  - memory paging, 151
  - memory size, 145
  - mode switch delay , 154
  - modes available, 195
  - modes supported, 151
  - NEC PC-9800, 80
  - output, 131
  - palette, 188
  - Paradise extensions, 140
  - parameter tables, 137
  - parameters, 132
  - PCjr, 119
  - PEL mask, 125
  - PEL mask, 126
  - preserving state, 135
  - reading characters, 120
  - registers, 193
  - relocated services, 138
  - resolution, 132
  - retrace frequency, 196
  - state flags, 134
  - screen resolution, 148
  - screen size, 146
  - screen updates, 824
  - screen address, 998
  - scrolling, 909
  - set display mode, 145
  - set emulation mode, 146
  - SOLLEX SuperVGA Extensions, 188
  - state information, 133
  - state mask, 146
  - SVGA state, 146
  - Texas Instruments PC, 179
  - touch screen, 181
  - Trident, 197
  - Vega installation check, 147
  - VGA, 138
  - Video7 installation check, 147
  - write BIOS string, 145
  - XGA, 136
  - zooming, 199
- Video7, 148
- VIDRAM, 1107
- VIDRAM;
  - entry point, 1107
- VIDRAMEGA, 758
- virtual memory, 24
- virtual interrupt flag, 674
- virtual machines, 809
- virtual devices;
  - entry points, 810
- virtual machines, 811
- virtual devices;
  - capabilities, 812
- virtual devices, 813
- Virtual DMA Specification -- see VDS, 365
  - session title, 623
  - session title, 623

  title switch, 622
Virtual Control Program Interface—see
    VCPI, 649
Virtual Interval Timer, 816
VirtualHDIRQ, 765
virus monitoring, 595
viruses, 1067
  installation check, 1072
  messages, 1072
  original INT 21h, 1072
  relocating, 1071
  resident code segment, 1071
  ROM INT 13h, 1073
  see also anti-viral tools, 1074
  user INT 13h, 1073
  viral code, 1073
VMiX, 901
VMiX;
  address mapping, 902
  banner window, 908
  BitBLT, 909
  built-in shell, 907
  channel control block, 904
  Channel Object manager, 901
  child processes, 906
  clearing window, 907
  colors, 908
  console input, 903
  copying memory, 909
  cursor position, 908
  executing protected-mode code, 910
  extended memory, 902
  fonts, 909
  GDT, 902
  initialization, 910
  installation check, 901
  invoking interrupts, 908
  IRP structure, 901
  job shell, 906
  memory allocation, 902
  Memory Object manager, 902
  object control block, 904
  page tables, 909
  process identifier, 904
  process termination, 906
  process control block, 904
  queue identifiers, 904
  queues, 904
  root window, 908
  screen output, 904
  scrolling display, 909
  serial I/O, 901
  suspending processes, 907
  System Configuration, 905
  TSS stack, 906
  VPGE structure, 902
VNETBIOS driver, 814
VNETBIOS driver;
  action codes, 814
volume number, 331
volume label, 401
VROOMM, 1021
VSAFE;
  uninstall, 1075
VSTACK, 278
VSWAP, 1167
  executing programs, 1167
  installation check, 1167
  resident code segment, 1167
VTD virtual device API, 811
VUIMAGE display driver;
  installation check, 163
VWATCH;
  uninstall, 1075
VxD, 813
  API, 807
  capabilities, 812
  DOSMGR, 814
  entry point, 810
  identifier, 810
  reference data, 807
  V86MMGR, 813
  VITD.386, 816
  VMD, 813
  VNETBIOS, 814
  VTD, 811
  After Dark, 289

Windows—see MS Windows, 805
WINOLDAP, 817
WINOLDAP;
  clipboard, 818
  device capabilities, 819
  device classification, 820
  GDI information, 819
  installation check, 817
  version check, 817
WinOS2, 624
WINX, 821
WINX;
  executing commands, 822
  installation check, 821
  server buffer, 821
  working directory, 821
Word Perfect, 1140
writing files, 398
  via FCB, 508
writing files, 520
X Windows, 874
  display name, 878
X00 FOSSIL;
  serial input, 261
  port control, 260
  serial output, 261
  status, 260
  initialization, 259
  port control, 259
XBDA, 46
XDI drivers;
  DVTXDI, 884
  DVSIXDI, 884
  DVXMS.DVR, 885
XDI, 879
XDI drivers;
  DVeop, 883
  QDPMI.SYS, 1108
  DESQview initialization, 879
  DESQview termination, 880
  driver custom subfunction, 879
  DV initialization, 882
  DVP processing, 881
  DVP processing, 882
  installation checks, 879
  keyboard focus, 881
  processes, 880
  save/restore state, 881
  swapping processes, 882
  XMS driver XDI structure, 880
xDISK;
  configuration, 219
  data buffer, 219
  installation check, 219
  undetermined function, 219
XGA, 138
XIOS functions, 920
XLOAD;
  installation check, 792
  memory size, 792
  resident code segment, 792
  undetermined functions, 792
XMA2EMS, 640
  installation check, 641
XMS, 737
XMS, 746
XMS;
  A20 handler, 798
  4DOS, 646
  RM386, 792
  calls from Cloaked applications, 686
  driver address, 747
  driver address, 747
  entry point, 747
  error codes, 751
  handle information, 784
  handles, 798
  installation check, 747
  RM386 extensions, 784
XMS2EMS;
  hidden frames, 641
XON/XOFF, 256
XpressFax, 1122
  installation check, 1122
XVIEW, 1168
XVIEW;
  error codes, 1168
  installation check, 1168
  page numbers, 1169
  popping down, 1169
year length, 592
YTERM, 1177
Z-Nix mouse driver;
  installation check, 464
Z-Nix mouse driver;
  see also mouse, 464
Z-Nix mouse driver;
  signature string, 464
Z100, 96
  keyboard, 97
  light pen, 97
  master 8259, 97
  parity error, 97
  printer interrupt, 97
  processor swap, 97
  rebooting, 98
  S100 vectored lines, 97
  serial ports, 97
  slave 8259, 97
  warm boot, 98
Zen Wu, 463
Zenith AT BIOS, 98
Zenith;
  *see also Z100*, 96
  AT BIOS, 98
  ROM debugger, 1054
ZFAX, 1124
  installation check, 1124
  uninstall, 1124
ZIPcodes, 1082
ZIPcodes, 1081
ZIPKEY, 1081
  area codes, 1084
  hotkeys, 1084
  installation check, 1081
  lookups, 1083
  save/restore context, 1083
  state codes, 1081
  stuffing keys, 1082
  version check, 1081
zoom window, 490
zooming display, 199
zooming;

PC PROGRAMMING

# PC INTERRUPTS

## SECOND EDITION

## The ultimate reference every PC programmer must have!

The first edition of *PC Interrupts* was the first and only complete reference to all the system calls an IBM programmer needed. Now updated and expanded, *PC Interrupts, Second Edition* provides concise descriptions of all the system calls from many different sources—MS-DOS, the ROM BIOS, and various APIs (application program interfaces) such as Windows™ 3.x, DESQview, and Advanced Power Management. Over 50 major APIs, dozens of resident utilities, as well as BIOS and MS-DOS services are covered. In addition, this book is the only available source of information on potential conflicts between calls from different APIs.

This book includes complete coverage of the following interrupt services and topics:

- ROM BIOS
- Multitaskers/task-switchers
- VCPI, DPMI, and DOS extenders
- Virtual DMA Specification
- Remote-control software
- FAX software
- Hardware and video
- Low-level disk and serial I/O
- Microsoft Windows
- Advanced Power Management
- Debugging tools
- Programming language support

A companion volume, *Network Interrupts: A Programmer's Reference to Network APIs* draws together all the information about network system calls, including NetWare 4, NetWare Lite, Windows for Workgroups, and Windows NT.

**Ralf Brown** is an independent software developer and author who recently received a Ph.D. in Computer Science from Carnegie-Mellon University. He maintains the well-known "Interrupt List" and coauthored *Undocumented DOS* (Addison-Wesley, 1993).

**Jim Kyle** is Director of research projects and co-founder of Automation Resources, Inc., and contributing editor/columnist for *Windows Tech Journal*. He also coauthored *Undocumented DOS* (Addison-Wesley 1993).

53995

9 780201 624854

ISBN 0-201-62485-0

ADDISON-WESLEY PUBLISHING COMPANY

**$39.95 US**
$51.95 CANADA